Advanced Machine Learning (BA-64061-001)

Assignment 3 - Convolution

Sgudise@kent.edu

GitHub: GitHub Link

Note - The professor provided Cats and Dogs Small, the dataset used for this project.

To assess the effects of changes, a baseline model was created, allowing for the assessment of both performance improvements and possible losses.

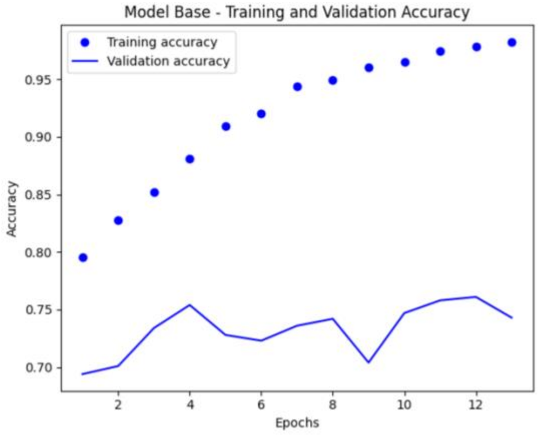
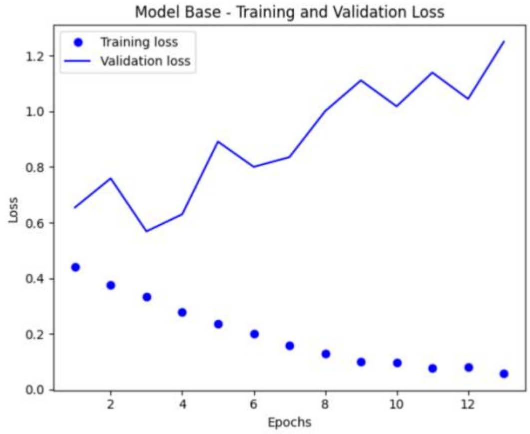
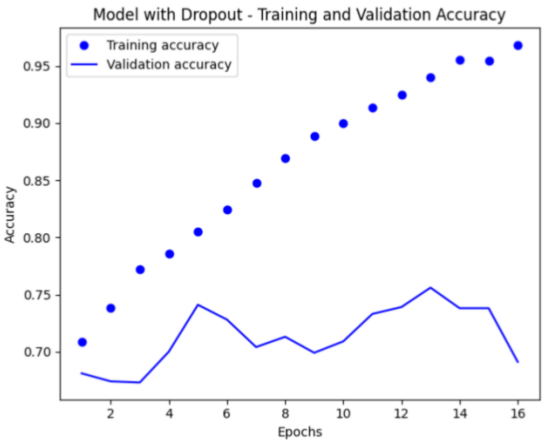
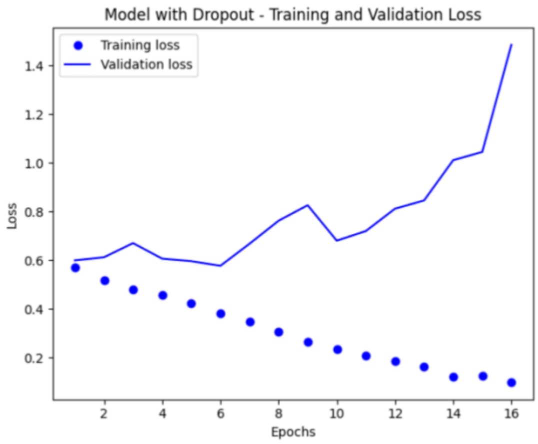
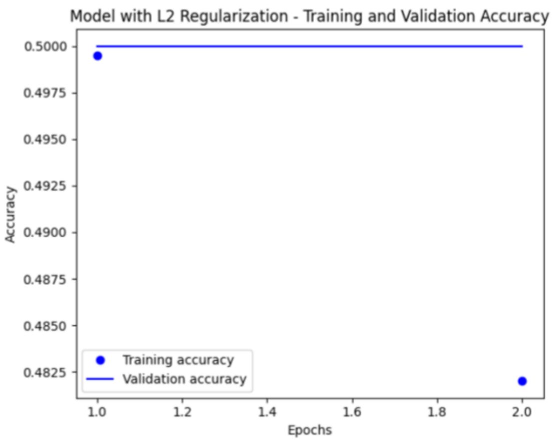
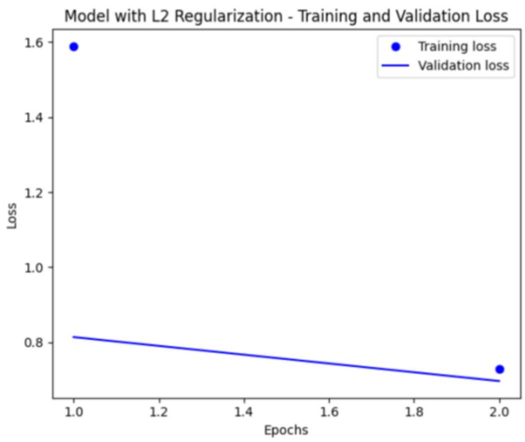
1. The following approach was implemented to reduce overfitting and adjust accuracy.

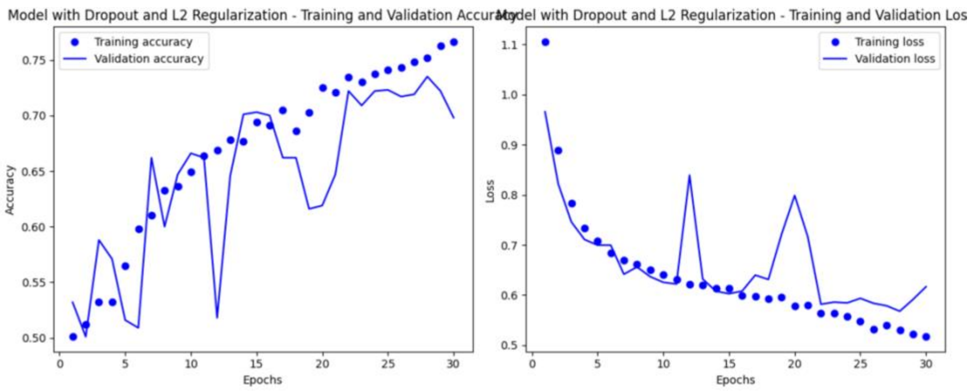
Model	**Training Image count**	**Methods**	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
Base	1000	none	0.95	0.76	0.73
Dropout	1000	Dropout 0.5	0.97	0.69	0.74
L2	1000	L2	0.48	0.5	0.52
Dropout & L2	1000	Dropout 0.25 & L2	0.77	0.69	0.74

Despite implementing dropout and L2 regularization techniques, the model continued to exhibit overfitting. While the training accuracy was high, the validation accuracy remained lower, suggesting that the model struggled to generalize effectively to unseen data.

Although dropout and L2 regularization helped to some extent in reducing overfitting, they were not sufficient to achieve high test accuracy with the limited dataset. This indicates that a more complex model or a larger dataset may be required for improved performance.

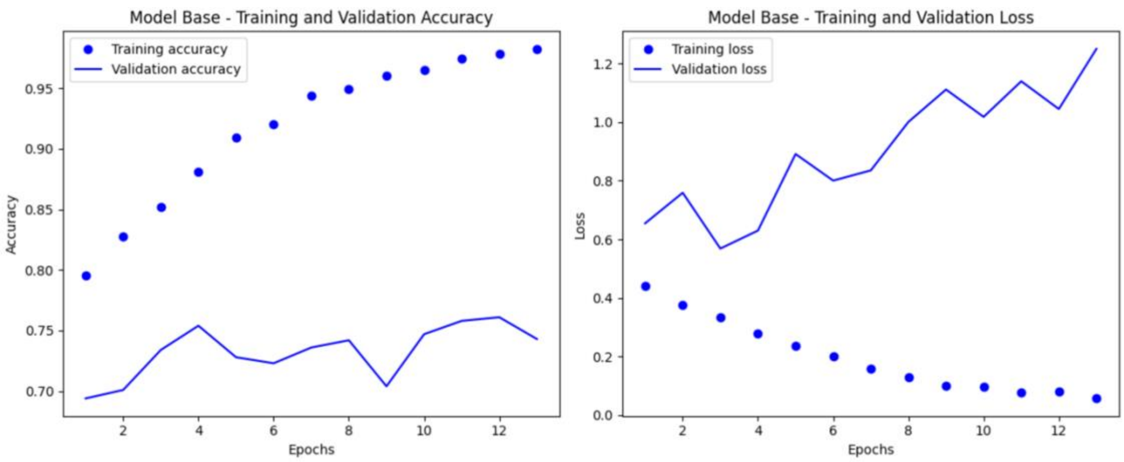
The best results were achieved with the **Dropout 0.5** model, where the accuracy was higher compared to the other configurations. Below are the graphs that illustrate the performance metrics of the various models tested.

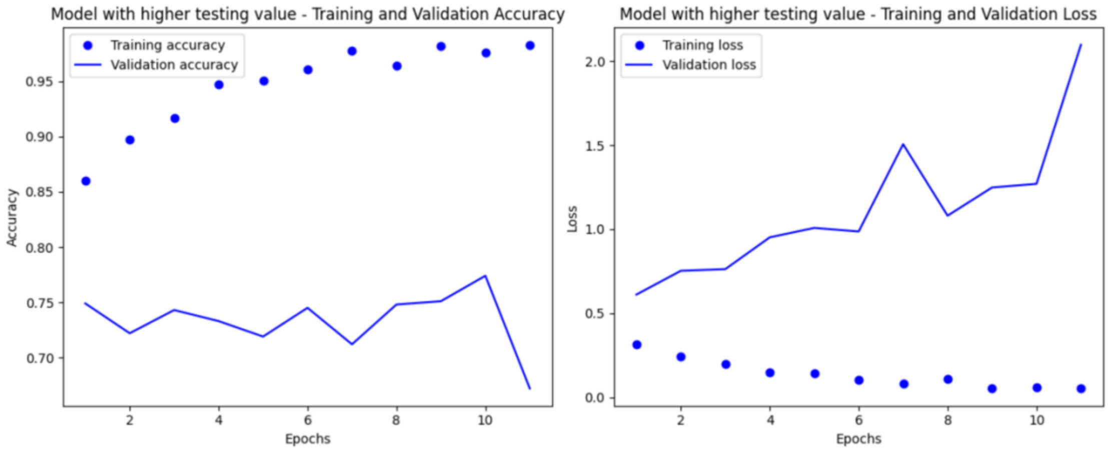

Model Base - Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs

Model Base - Training and Validation Loss
Training loss
Validation loss
Loss
Epochs

Model with Dropout - Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs

Model with Dropout - Training and Validation Loss
Training loss
Validation loss
Loss
Epochs

Model with L2 Regularization - Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs

Model with L2 Regularization - Training and Validation Loss
Training loss
Validation loss
Loss
Epochs

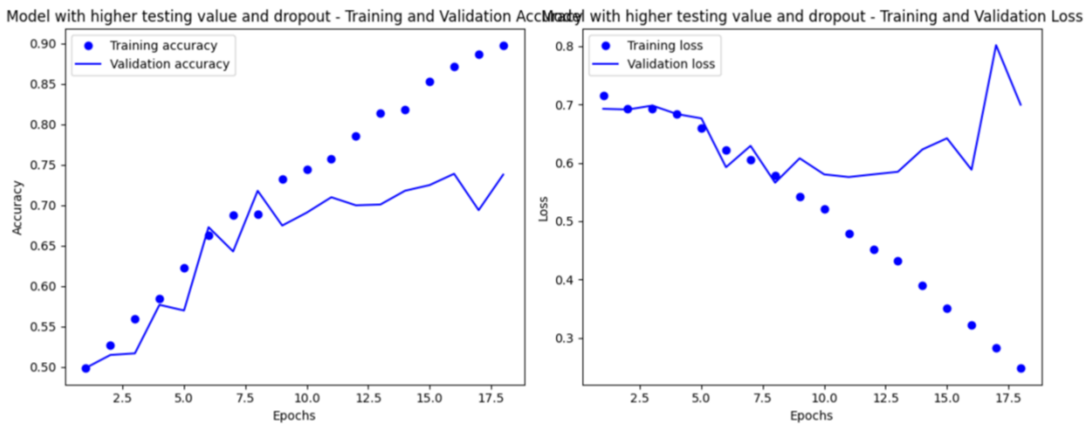


2&3. Since these questions are regrading training set size both the question was merged in one. The training set was increased by 50% (1000 to 1500) but the model.

Model	**Training Image count**	**Methods**	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
Base	1000	none	0.95	0.76	0.73
Model 1500	1500	none	0.97	0.67	0.72
Dropout and 2000	2000	Dropout	0.90	0.73	0.75







2. Model_base_1500 (The same Base Model is used with 1500 Training Images)

New Training Sample Size: 1500 images

Validation and Test Sample Sizes: 500 each (same as Step 1)

Model Configuration: A comparable CNN architecture was trained from scratch with no regularization techniques applied.

Objective: To evaluate whether increasing the training sample size improves model performance when training from scratch.

Results: Test accuracy: 0.72

Observations:

- Limited Improvement: The test accuracy remained at 0.72, which is close to the performance achieved with 1000 training samples. This suggests that increasing the training set size from 1000 to 1500 did not lead to significant improvement in the model’s ability to generalize to unseen data.

- Possible Causes: The lack of improvement may be due to the relatively simple architecture of the model, which may have already reached its capacity for learning meaningful features with the initial dataset size. Additionally, the extra 500 samples may have been too similar to the original data, contributing little in terms of feature diversity.

3. Model_d_2000 (Model with Dropout 0.25 and 2000 Training Images)

Adjusted Training Sample Size: 2000 images

Validation and Test Sample Sizes: 500 each

Model Configuration: The same CNN architecture, trained from scratch with dropout (0.25) applied as a regularization technique.

Objective: To determine if increasing the training sample size improves generalization and model accuracy.

Results: Test accuracy: 0.75

Observations:

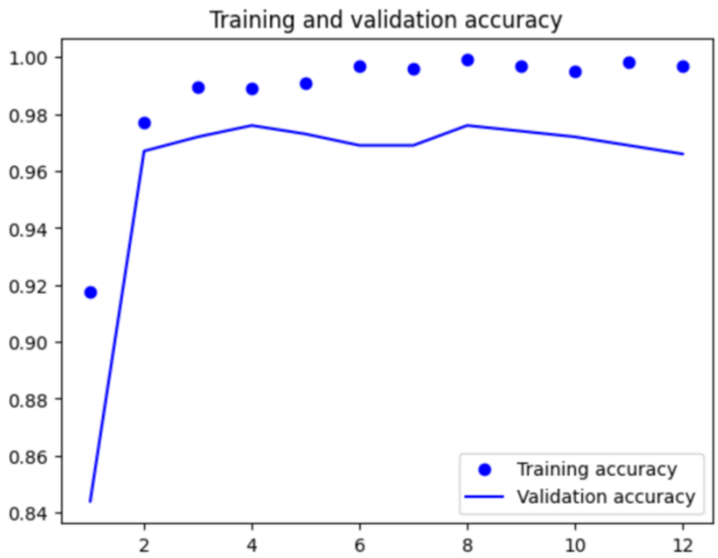
- Performance Improvement: Increasing the training sample size to 2000 resulted in a slight improvement in test accuracy (from 0.72 to 0.75). This suggests that the larger training set allowed the model to learn more diverse features, which helped improve generalization.

- Persistent Overfitting: Despite the improvement in accuracy, the model still exhibited signs of overfitting. Training accuracy was higher than validation accuracy, indicating that further regularization or a more complex model may be necessary to address this issue.

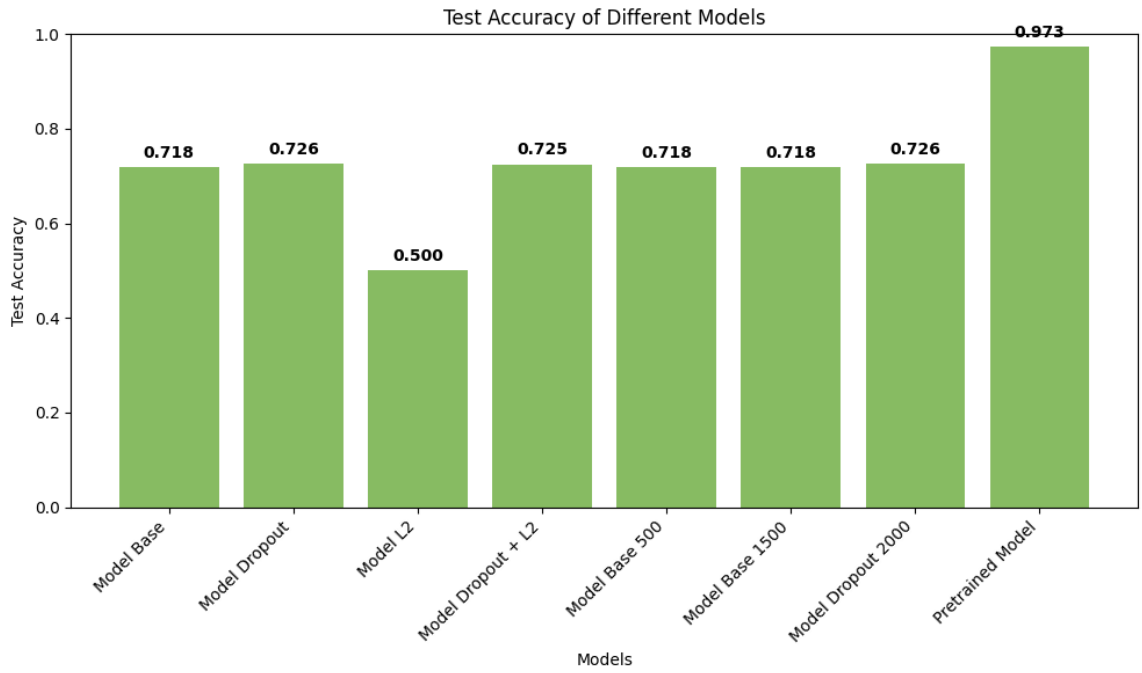
- Conclusion: A training sample size of 2000 provided the best performance for the model trained from scratch. However, further increases in training data might require a more sophisticated model architecture or additional regularization techniques to prevent overfitting.

4. Optimizing pre-trained model

In this section, a VGG16 model pre-trained on ImageNet was fine-tuned on the Cats & Dogs dataset using the same sample sizes as those used for training the model from scratch. This approach allows leveraging the pre-learned features from ImageNet to enhance performance on the target dataset.



Model	Training Image count	Methods	Training Accuracy	Validation Accuracy	Testing Accuracy
Base	1000	none	0.95	0.76	0.74
Dropout	1000	Dropout 0.5	0.97	0.69	0.74
L2	1000	L2	0.48	0.5	0.52
Dropout & L2	1000	Dropout 0.25 & L2	0.77	0.69	0.74
Model 1500	1500	none	0.97	0.67	0.72
Dropout and 2000	2000	Dropout	0.96	0.78	0.74
Pretrained	NA	Dropout	0.90	0.73	0.75



Results: Accuracy: 0.973

Observations:

- Superior Performance of Pre-trained Model: The pre-trained VGG16 model achieved the highest accuracy (0.973) when trained with 2000 samples. This demonstrates that pre-trained models can yield excellent performance even with limited data, and their accuracy improves further as the training sample size increases, particularly in fine-tuning tasks.
- Minimal Overfitting: The pre-trained model exhibited minimal overfitting, with training and validation accuracy closely aligned. This highlights the stability of pre-trained models in achieving high accuracy while minimizing the risk of overfitting, which is a common issue with models trained from scratch.
- Training Sample Size and Model Performance: When training models from scratch, increasing the training sample size from 1000 to 2000 resulted in a modest increase in accuracy (from 0.718 to 0.72). However, even with larger sample sets, models trained from scratch did not achieve performance levels comparable to those of the pre-trained model.
- Pretrained Models vs. Models Trained from Scratch: The pre-trained VGG16 model consistently outperformed the models trained from scratch, reaching a high accuracy of 0.973 with just 1000 training samples. This underscores the advantage of transfer

learning, where pre-trained models leverage existing feature representations to achieve outstanding results even with limited data.

Recommendations:

For Small to Moderate Datasets: Pre-trained models should be prioritized due to their significant performance advantages. They are particularly effective when the dataset is small to moderate in size, as they require fewer samples to deliver high accuracy. For Larger Datasets: Training from scratch may be more appropriate for larger datasets, especially when supported by robust regularization techniques and model modifications to prevent overfitting.

```
import os
import shutil
import pathlib

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

i_p = pathlib.Path(r"/content/drive/MyDrive/cats_vs_dogs_small")
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

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

x_base_input = keras.Input(shape=(180, 180, 3))
x_base_rescale = layers.Rescaling(1./255)(x_base_input)
x_base_conv1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x_base_rescale)
x_base_pool1 = layers.MaxPooling2D(pool_size=2)(x_base_conv1)
x_base_conv2 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x_base_pool1)
x_base_pool2 = layers.MaxPooling2D(pool_size=2)(x_base_conv2)
x_base_conv3 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x_base_pool2)
x_base_pool3 = layers.MaxPooling2D(pool_size=2)(x_base_conv3)
x_base_conv4 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x_base_pool3)
x_base_pool4 = layers.MaxPooling2D(pool_size=2)(x_base_conv4)
x_base_conv5 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x_base_pool4)
x_base_flatten = layers.Flatten()(x_base_conv5)
x_base_output = layers.Dense(1, activation="sigmoid")(x_base_flatten)

mod_base = keras.Model(inputs=x_base_input, outputs=x_base_output)
mod_base_1500 = keras.Model(inputs=x_base_input, outputs=x_base_output)
mod_base_500 = keras.Model(inputs=x_base_input, outputs=x_base_output)
```

Model With dropout

```
from tensorflow import keras
from tensorflow.keras import layers, regularizers

# Base input
x_d_input = keras.Input(shape=(180, 180, 3))
x_d_rescale = layers.Rescaling(1./255)(x_d_input)
x_d_conv1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x_d_rescale)
x_d_pool1 = layers.MaxPooling2D(pool_size=2)(x_d_conv1)
x_d_conv2 = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x_d_pool1)
x_d_pool2 = layers.MaxPooling2D(pool_size=2)(x_d_conv2)
x_d_conv3 = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x_d_pool2)
x_d_pool3 = layers.MaxPooling2D(pool_size=2)(x_d_conv3)
x_d_conv4 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x_d_pool3)
x_d_pool4 = layers.MaxPooling2D(pool_size=2)(x_d_conv4)
x_d_conv5 = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x_d_pool4)
x_d_pool5 = layers.Flatten()(x_d_conv5)
```

```
x_d_dropout = layers.Dropout(0.5)(x_d_pool5)
x_d_output = layers.Dense(1, activation="sigmoid")(x_d_dropout)

mod_d = keras.Model(inputs=x_d_input, outputs=x_d_output)
mod_d_2000 = keras.Model(inputs=x_d_input, outputs=x_d_output)
```

Model with L2

```
from tensorflow import keras
from tensorflow.keras import layers, regularizers

x_L2_input = keras.Input(shape=(180, 180, 3))
x_L2_rescale = layers.Rescaling(1./255)(x_L2_input)
x_L2_conv1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.005))(x_L2_rescale)
x_L2_pool1 = layers.MaxPooling2D(pool_size=2)(x_L2_conv1)
x_L2_conv2 = layers.Conv2D(filters=64, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.005))(x_L2_pool1)
x_L2_pool2 = layers.MaxPooling2D(pool_size=2)(x_L2_conv2)
x_L2_conv3 = layers.Conv2D(filters=128, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.005))(x_L2_pool2)
x_L2_pool3 = layers.MaxPooling2D(pool_size=2)(x_L2_conv3)
x_L2_conv4 = layers.Conv2D(filters=256, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.005))(x_L2_pool3)
x_L2_pool4 = layers.MaxPooling2D(pool_size=2)(x_L2_conv4)
x_L2_conv5 = layers.Conv2D(filters=256, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.005))(x_L2_pool4)
x_L2_flatten = layers.Flatten()(x_L2_conv5)
x_L2_output = layers.Dense(1, activation="sigmoid")(x_L2_flatten)

mod_L2 = keras.Model(inputs=x_L2_input, outputs=x_L2_output)
```

Model with Dropout and L2

```
from tensorflow import keras
from tensorflow.keras import layers, regularizers

x_d_L2_input = keras.Input(shape=(180, 180, 3))
x_d_L2_rescale = layers.Rescaling(1./255)(x_d_L2_input)
x_d_L2_conv1 = layers.Conv2D(filters=32, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.001))(x_d_L2_rescale)
x_d_L2_pool1 = layers.MaxPooling2D(pool_size=2)(x_d_L2_conv1)
x_d_L2_conv2 = layers.Conv2D(filters=64, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.001))(x_d_L2_pool1)
x_d_L2_pool2 = layers.MaxPooling2D(pool_size=2)(x_d_L2_conv2)
x_d_L2_conv3 = layers.Conv2D(filters=128, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.001))(x_d_L2_pool2)
x_d_L2_pool3 = layers.MaxPooling2D(pool_size=2)(x_d_L2_conv3)
x_d_L2_conv4 = layers.Conv2D(filters=256, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.001))(x_d_L2_pool3)
x_d_L2_pool4 = layers.MaxPooling2D(pool_size=2)(x_d_L2_conv4)
x_d_L2_conv5 = layers.Conv2D(filters=256, kernel_size=3, activation="relu", kernel_regularizer=regularizers.l2(0.001))(x_d_L2_pool4)
x_d_L2_flatten = layers.Flatten()(x_d_L2_conv5)
x_d_L2_dropout = layers.Dropout(0.25)(x_d_L2_flatten)
x_d_L2_output = layers.Dense(1, activation="sigmoid")(x_d_L2_dropout)

mod_d_L2 = keras.Model(inputs=x_d_L2_input, outputs=x_d_L2_output)
```

```
mod_base.summary()
```

Model: "functional"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_1 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_2 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_3 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_3 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_4 (Conv2D)	(None, 7, 7, 256)	590,080
flatten (Flatten)	(None, 12544)	0
dense (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)

```
mod_d.summary()
```

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
dropout (Dropout)	(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)

```
mod_L2.summary()
```

Model: "functional_5"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 180, 180, 3)	0
rescaling_2 (Rescaling)	(None, 180, 180, 3)	0
conv2d_10 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_11 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_12 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_10 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_13 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_11 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_14 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_2 (Flatten)	(None, 12544)	0
dense_2 (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)

```
mod_d_L2.summary()
```

Model: "functional_6"

Layer (type)	Output Shape	Param #
input_layer_3 (InputLayer)	(None, 180, 180, 3)	0
rescaling_3 (Rescaling)	(None, 180, 180, 3)	0
conv2d_15 (Conv2D)	(None, 178, 178, 32)	896
max_pooling2d_12 (MaxPooling2D)	(None, 89, 89, 32)	0
conv2d_16 (Conv2D)	(None, 87, 87, 64)	18,496
max_pooling2d_13 (MaxPooling2D)	(None, 43, 43, 64)	0
conv2d_17 (Conv2D)	(None, 41, 41, 128)	73,856
max_pooling2d_14 (MaxPooling2D)	(None, 20, 20, 128)	0
conv2d_18 (Conv2D)	(None, 18, 18, 256)	295,168
max_pooling2d_15 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_19 (Conv2D)	(None, 7, 7, 256)	590,080
flatten_3 (Flatten)	(None, 12544)	0
dropout_1 (Dropout)	(None, 12544)	0
dense_3 (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)

Setting up every model for training

assembling every Keras model into TensorFlow-SMD

To modify, double-click (or press Enter).

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

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

```
mod_L2.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
mod_d_L2.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
```

```
              metrics=["accuracy"])


mod_base_1500.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


mod_d_2000.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


mod_base_500.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])


import tensorflow as tf
from tensorflow.keras.utils import image_dataset_from_directory

# Set the random seed for reproducibility
seed = 143
tf.random.set_seed(seed)

# Load the datasets from the directory with shuffling
train_full_dataset = image_dataset_from_directory(
    i_p / "train",
    image_size=(180, 180),
    batch_size=32,
    shuffle=True,
    seed=seed
)

validation_full_dataset = image_dataset_from_directory(
    i_p / "validation",
    image_size=(180, 180),
    batch_size=32,
    shuffle=True,
    seed=seed
)

test_full_dataset = image_dataset_from_directory(
    i_p / "test",
    image_size=(180, 180),
    batch_size=32,
    shuffle=True,
    seed=seed
)

# Create smaller datasets
train_dataset = train_full_dataset.take(1000)
train_dataset_1500 = train_full_dataset.take(1500)
train_dataset_500 = train_full_dataset.take(500)
train_dataset_2000 = train_dataset.shuffle(buffer_size=2000)
validation_dataset = validation_full_dataset.take(500)
```

```
validation_dataset_1000 = validation_full_dataset.take(1000)
test_dataset = test_full_dataset.take(500)
```

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

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

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

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

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

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

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

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

```
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,)
```

Adjusting the model to the data

```
# Define the callbacks for model training
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss"),
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=10,  # Number of epochs to wait for improvement
        restore_best_weights=True  # Restore model weights from the epoch with the best value
    )
]


# Fit the base model
hist_base = mod_base.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 264s 4s/step - accuracy: 0.5283 - loss: 0.7246 - val_accuracy: 0.4990 - val_loss: 0.8923
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 135ms/step - accuracy: 0.5302 - loss: 0.7050 - val_accuracy: 0.6200 - val_loss: 0.6730
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.5621 - loss: 0.6812 - val_accuracy: 0.6260 - val_loss: 0.6528
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.6180 - loss: 0.6628 - val_accuracy: 0.6120 - val_loss: 0.6511
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 177ms/step - accuracy: 0.6453 - loss: 0.6164 - val_accuracy: 0.6600 - val_loss: 0.6036
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 135ms/step - accuracy: 0.6817 - loss: 0.5980 - val_accuracy: 0.5070 - val_loss: 0.8728
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.6702 - loss: 0.5989 - val_accuracy: 0.7130 - val_loss: 0.5693
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.7468 - loss: 0.5291 - val_accuracy: 0.7180 - val_loss: 0.5454
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 142ms/step - accuracy: 0.7561 - loss: 0.5123 - val_accuracy: 0.7370 - val_loss: 0.5470
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 133ms/step - accuracy: 0.7757 - loss: 0.4735 - val_accuracy: 0.7280 - val_loss: 0.5236
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 190ms/step - accuracy: 0.7996 - loss: 0.4237 - val_accuracy: 0.7250 - val_loss: 0.5868
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.8277 - loss: 0.3763 - val_accuracy: 0.7130 - val_loss: 0.5508
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 152ms/step - accuracy: 0.8425 - loss: 0.3514 - val_accuracy: 0.7400 - val_loss: 0.6015
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 175ms/step - accuracy: 0.8567 - loss: 0.3086 - val_accuracy: 0.7390 - val_loss: 0.6370
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 184ms/step - accuracy: 0.8881 - loss: 0.2548 - val_accuracy: 0.7270 - val_loss: 0.6593
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 146ms/step - accuracy: 0.9169 - loss: 0.1960 - val_accuracy: 0.7290 - val_loss: 0.9468
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 158ms/step - accuracy: 0.9378 - loss: 0.1806 - val_accuracy: 0.7440 - val_loss: 0.8899
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9497 - loss: 0.1444 - val_accuracy: 0.7360 - val_loss: 1.0322
```

```
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.9634 - loss: 0.1069 - val_accuracy: 0.6970 - val_loss: 1.1594
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 137ms/step - accuracy: 0.9546 - loss: 0.1183 - val_accuracy: 0.7650 - val_loss: 1.0166
```

```
# Fit the base model
hist_base = mod_base_500.fit(
    train_dataset_500,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 153ms/step - accuracy: 0.7837 - loss: 0.4707 - val_accuracy: 0.6940 - val_loss: 0.6546
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.8279 - loss: 0.3926 - val_accuracy: 0.7010 - val_loss: 0.7588
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.8509 - loss: 0.3360 - val_accuracy: 0.7340 - val_loss: 0.5685
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 177ms/step - accuracy: 0.8825 - loss: 0.2742 - val_accuracy: 0.7540 - val_loss: 0.6299
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 138ms/step - accuracy: 0.9137 - loss: 0.2345 - val_accuracy: 0.7280 - val_loss: 0.8910
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9098 - loss: 0.2148 - val_accuracy: 0.7230 - val_loss: 0.8002
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9426 - loss: 0.1584 - val_accuracy: 0.7360 - val_loss: 0.8351
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.9489 - loss: 0.1253 - val_accuracy: 0.7420 - val_loss: 1.0005
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 140ms/step - accuracy: 0.9509 - loss: 0.1152 - val_accuracy: 0.7040 - val_loss: 1.1110
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9572 - loss: 0.1184 - val_accuracy: 0.7470 - val_loss: 1.0176
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.9727 - loss: 0.0724 - val_accuracy: 0.7580 - val_loss: 1.1391
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9823 - loss: 0.0581 - val_accuracy: 0.7610 - val_loss: 1.0444
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 132ms/step - accuracy: 0.9799 - loss: 0.0649 - val_accuracy: 0.7430 - val_loss: 1.2499
```

```
# Fit the base model
hist_base_1500 = mod_base_1500.fit(
    train_dataset_1500,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 162ms/step - accuracy: 0.8519 - loss: 0.3590 - val_accuracy: 0.7490 - val_loss: 0.6105
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.8872 - loss: 0.2624 - val_accuracy: 0.7220 - val_loss: 0.7522
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 194ms/step - accuracy: 0.9131 - loss: 0.2034 - val_accuracy: 0.7430 - val_loss: 0.7620
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.9468 - loss: 0.1517 - val_accuracy: 0.7330 - val_loss: 0.9513
Epoch 5/30
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 158ms/step - accuracy: 0.9425 - loss: 0.1731 - val_accuracy: 0.7190 - val_loss: 1.0075
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.9501 - loss: 0.1209 - val_accuracy: 0.7450 - val_loss: 0.9861
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9727 - loss: 0.0984 - val_accuracy: 0.7120 - val_loss: 1.5060
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.9517 - loss: 0.1373 - val_accuracy: 0.7480 - val_loss: 1.0804
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 178ms/step - accuracy: 0.9856 - loss: 0.0456 - val_accuracy: 0.7510 - val_loss: 1.2483
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 149ms/step - accuracy: 0.9714 - loss: 0.0712 - val_accuracy: 0.7740 - val_loss: 1.2699
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 185ms/step - accuracy: 0.9753 - loss: 0.0721 - val_accuracy: 0.6720 - val_loss: 2.0982
```

```
# Fit the base model
hist_d_2000 = mod_d_2000.fit(
    train_dataset_2000,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 17s 128ms/step - accuracy: 0.4931 - loss: 0.7692 - val_accuracy: 0.4990 - val_loss: 0.6925
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 88ms/step - accuracy: 0.5386 - loss: 0.6907 - val_accuracy: 0.5150 - val_loss: 0.6913
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 91ms/step - accuracy: 0.5569 - loss: 0.6928 - val_accuracy: 0.5170 - val_loss: 0.6981
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 115ms/step - accuracy: 0.5783 - loss: 0.6991 - val_accuracy: 0.5770 - val_loss: 0.6835
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 17s 84ms/step - accuracy: 0.6021 - loss: 0.6655 - val_accuracy: 0.5700 - val_loss: 0.6760
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 115ms/step - accuracy: 0.6545 - loss: 0.6260 - val_accuracy: 0.6730 - val_loss: 0.5924
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 79ms/step - accuracy: 0.6963 - loss: 0.5781 - val_accuracy: 0.6430 - val_loss: 0.6290
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 116ms/step - accuracy: 0.6859 - loss: 0.5726 - val_accuracy: 0.7180 - val_loss: 0.5658
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 116ms/step - accuracy: 0.7343 - loss: 0.5356 - val_accuracy: 0.6750 - val_loss: 0.6077
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 89ms/step - accuracy: 0.7511 - loss: 0.5071 - val_accuracy: 0.6910 - val_loss: 0.5801
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 98ms/step - accuracy: 0.7618 - loss: 0.4823 - val_accuracy: 0.7100 - val_loss: 0.5756
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 95ms/step - accuracy: 0.7859 - loss: 0.4419 - val_accuracy: 0.7000 - val_loss: 0.5801
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 78ms/step - accuracy: 0.8092 - loss: 0.4355 - val_accuracy: 0.7010 - val_loss: 0.5846
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 79ms/step - accuracy: 0.8251 - loss: 0.3813 - val_accuracy: 0.7180 - val_loss: 0.6229
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 116ms/step - accuracy: 0.8605 - loss: 0.3234 - val_accuracy: 0.7250 - val_loss: 0.6420
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 116ms/step - accuracy: 0.8776 - loss: 0.3086 - val_accuracy: 0.7390 - val_loss: 0.5881
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 115ms/step - accuracy: 0.8878 - loss: 0.2878 - val_accuracy: 0.6940 - val_loss: 0.8016
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 115ms/step - accuracy: 0.9039 - loss: 0.2332 - val_accuracy: 0.7380 - val_loss: 0.6995
```

```
# Repeat for other models
hist_d = mod_d.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 170ms/step - accuracy: 0.7134 - loss: 0.5830 - val_accuracy: 0.6810 - val_loss: 0.5996
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 197ms/step - accuracy: 0.7460 - loss: 0.5139 - val_accuracy: 0.6740 - val_loss: 0.6124
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 165ms/step - accuracy: 0.7707 - loss: 0.4906 - val_accuracy: 0.6730 - val_loss: 0.6705
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.7666 - loss: 0.4792 - val_accuracy: 0.7000 - val_loss: 0.6067
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.7985 - loss: 0.4309 - val_accuracy: 0.7410 - val_loss: 0.5963
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 143ms/step - accuracy: 0.8166 - loss: 0.3909 - val_accuracy: 0.7280 - val_loss: 0.5773
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.8428 - loss: 0.3486 - val_accuracy: 0.7040 - val_loss: 0.6678
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 158ms/step - accuracy: 0.8572 - loss: 0.3261 - val_accuracy: 0.7130 - val_loss: 0.7623
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.8870 - loss: 0.2684 - val_accuracy: 0.6990 - val_loss: 0.8261
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 142ms/step - accuracy: 0.8948 - loss: 0.2401 - val_accuracy: 0.7090 - val_loss: 0.6805
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 157ms/step - accuracy: 0.9074 - loss: 0.2228 - val_accuracy: 0.7330 - val_loss: 0.7201
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 163ms/step - accuracy: 0.9254 - loss: 0.1984 - val_accuracy: 0.7390 - val_loss: 0.8119
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.9455 - loss: 0.1638 - val_accuracy: 0.7560 - val_loss: 0.8457
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.9557 - loss: 0.1288 - val_accuracy: 0.7380 - val_loss: 1.0110
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 165ms/step - accuracy: 0.9551 - loss: 0.1199 - val_accuracy: 0.7380 - val_loss: 1.0448
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 163ms/step - accuracy: 0.9715 - loss: 0.0970 - val_accuracy: 0.6910 - val_loss: 1.4847
```

```
hist_L2 = mod_L2.fit(
    train_dataset,
    epochs=2, # Best so stopping at 2
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/2
63/63 ━━━━━━━━━━━━━━━━━━━━ 16s 207ms/step - accuracy: 0.5043 - loss: 2.2580 - val_accuracy: 0.5000 - val_loss: 0.8140
Epoch 2/2
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 162ms/step - accuracy: 0.4836 - loss: 0.7579 - val_accuracy: 0.5000 - val_loss: 0.6970
```

```
hist_d_L2 = mod_d_L2.fit(
    train_dataset,
    epochs=30,
```

```
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.5194 - loss: 0.9256 - val_accuracy: 0.5010 - val_loss: 0.8222
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.5227 - loss: 0.8012 - val_accuracy: 0.5880 - val_loss: 0.7461
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 138ms/step - accuracy: 0.5169 - loss: 0.7440 - val_accuracy: 0.5710 - val_loss: 0.7110
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 194ms/step - accuracy: 0.5654 - loss: 0.7118 - val_accuracy: 0.5160 - val_loss: 0.6996
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.5860 - loss: 0.6901 - val_accuracy: 0.5090 - val_loss: 0.6996
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 177ms/step - accuracy: 0.5851 - loss: 0.6830 - val_accuracy: 0.6620 - val_loss: 0.6416
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 138ms/step - accuracy: 0.6219 - loss: 0.6691 - val_accuracy: 0.6000 - val_loss: 0.6561
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 143ms/step - accuracy: 0.6242 - loss: 0.6529 - val_accuracy: 0.6470 - val_loss: 0.6369
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.6367 - loss: 0.6480 - val_accuracy: 0.6660 - val_loss: 0.6256
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 204ms/step - accuracy: 0.6558 - loss: 0.6441 - val_accuracy: 0.6620 - val_loss: 0.6221
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 165ms/step - accuracy: 0.6585 - loss: 0.6220 - val_accuracy: 0.5180 - val_loss: 0.8392
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 180ms/step - accuracy: 0.6672 - loss: 0.6319 - val_accuracy: 0.6460 - val_loss: 0.6322
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 142ms/step - accuracy: 0.6745 - loss: 0.6123 - val_accuracy: 0.7010 - val_loss: 0.6078
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 197ms/step - accuracy: 0.6831 - loss: 0.6366 - val_accuracy: 0.7030 - val_loss: 0.6028
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.6781 - loss: 0.6123 - val_accuracy: 0.7000 - val_loss: 0.6080
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.7168 - loss: 0.5987 - val_accuracy: 0.6620 - val_loss: 0.6399
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.7038 - loss: 0.5889 - val_accuracy: 0.6620 - val_loss: 0.6312
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.7150 - loss: 0.5865 - val_accuracy: 0.6160 - val_loss: 0.7210
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 202ms/step - accuracy: 0.7153 - loss: 0.5932 - val_accuracy: 0.6190 - val_loss: 0.7988
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.7211 - loss: 0.5934 - val_accuracy: 0.6470 - val_loss: 0.7164
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 182ms/step - accuracy: 0.7306 - loss: 0.5732 - val_accuracy: 0.7220 - val_loss: 0.5817
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.7430 - loss: 0.5541 - val_accuracy: 0.7090 - val_loss: 0.5860
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 174ms/step - accuracy: 0.7385 - loss: 0.5554 - val_accuracy: 0.7220 - val_loss: 0.5845
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 146ms/step - accuracy: 0.7433 - loss: 0.5341 - val_accuracy: 0.7230 - val_loss: 0.5935
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 194ms/step - accuracy: 0.7411 - loss: 0.5349 - val_accuracy: 0.7170 - val_loss: 0.5837
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.7468 - loss: 0.5353 - val_accuracy: 0.7190 - val_loss: 0.5787
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 162ms/step - accuracy: 0.7571 - loss: 0.5337 - val_accuracy: 0.7350 - val_loss: 0.5679
```

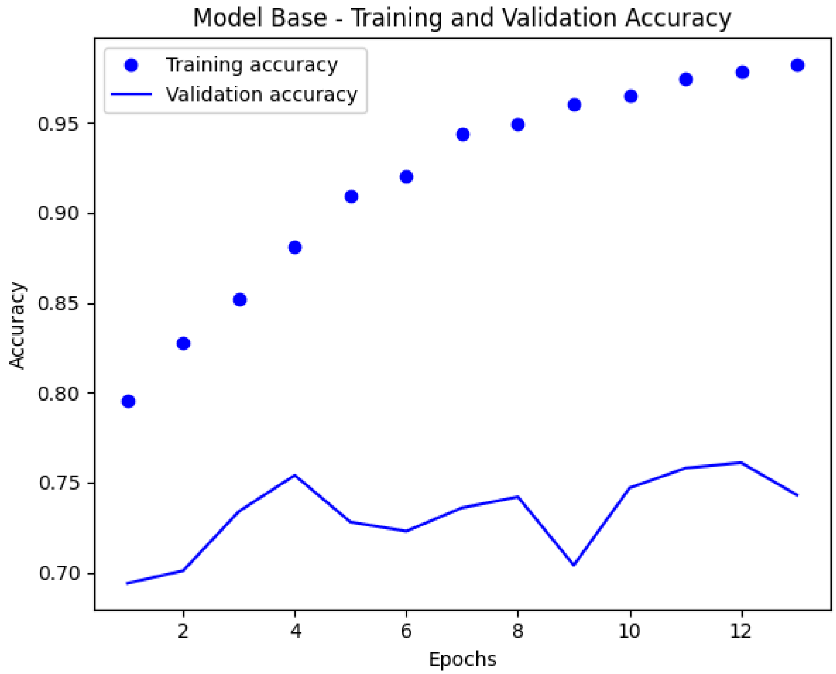
```
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 144ms/step - accuracy: 0.7727 - loss: 0.5221 - val_accuracy: 0.6980 - val_loss: 0.6167
```

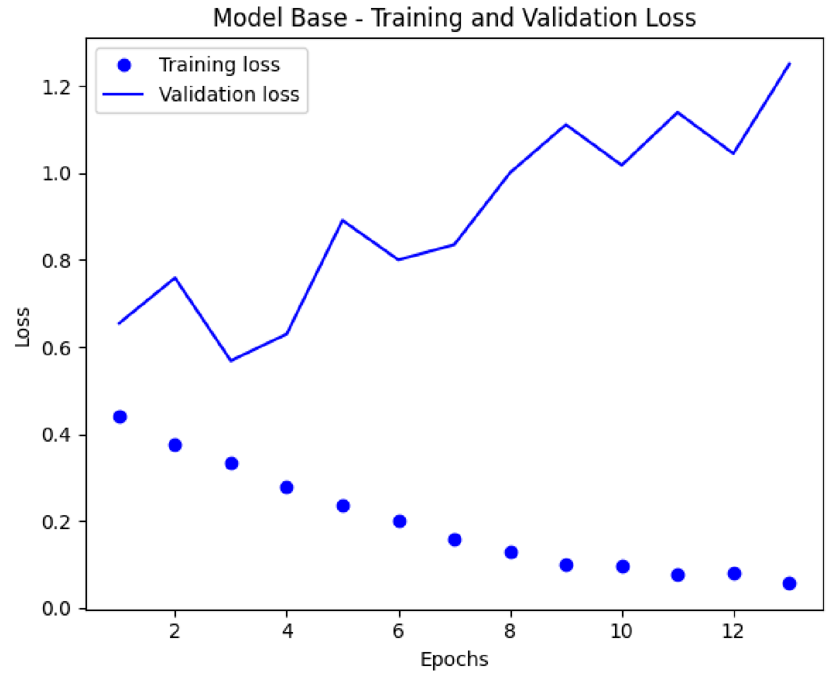
Question 2 Higher training value

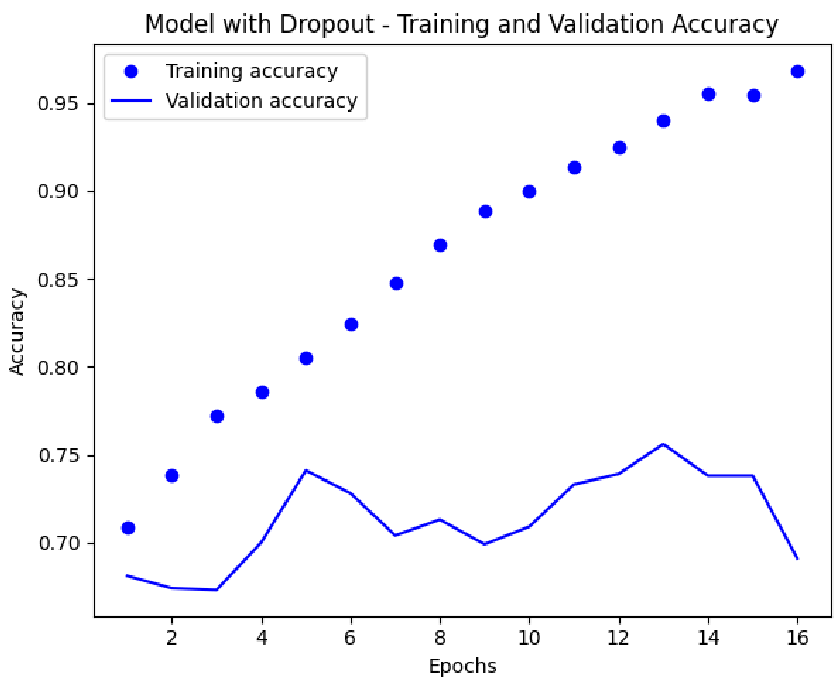
Question 3 Higher training value

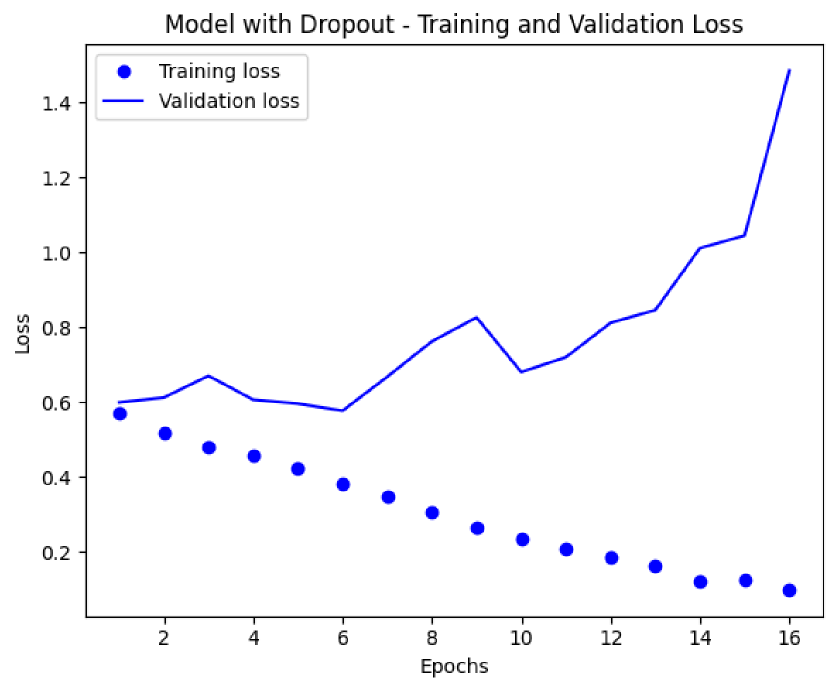
Displaying curves of loss and accuracy during training

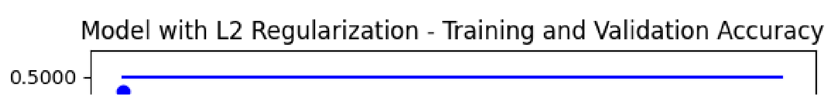
```
import matplotlib.pyplot as plt

# Function to plot training and validation metrics
def plot_training_history(history, model_name):
    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)

    # Plot accuracy
    plt.figure(figsize=(12, 5))  # Create a new figure for each model
    plt.subplot(1, 2, 1)  # Create a subplot for accuracy
    plt.plot(epochs, accuracy, "bo", label="Training accuracy")
    plt.plot(epochs, val_accuracy, "b", label="Validation accuracy")
    plt.title(f"{model_name} - Training and Validation Accuracy")
    plt.xlabel("Epochs")
    plt.ylabel("Accuracy")
    plt.legend()

    # Plot loss
    plt.subplot(1, 2, 2)  # Create a subplot for loss
    plt.plot(epochs, loss, "bo", label="Training loss")
    plt.plot(epochs, val_loss, "b", label="Validation loss")
    plt.title(f"{model_name} - Training and Validation Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()

    plt.tight_layout()  # Adjust the layout
    plt.show()

# Plot training history for each model
plot_training_history(hist_base, "Model Base")
plot_training_history(hist_d, "Model with Dropout")
plot_training_history(hist_L2, "Model with L2 Regularization")
plot_training_history(hist_d_L2, "Model with Dropout and L2 Regularization")
plot_training_history(hist_base_1500, "Model with higher testing value")
plot_training_history(hist_d_2000, "Model with higher testing value and dropout")
```

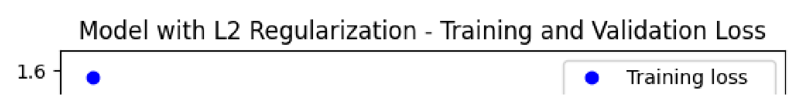

Model Base - Training and Validation Accuracy
Training accuracy
Validation accuracy
0.70
0.75
0.80
0.85
0.90
0.95
Accuracy
2
4
6
8
10
12
Epochs

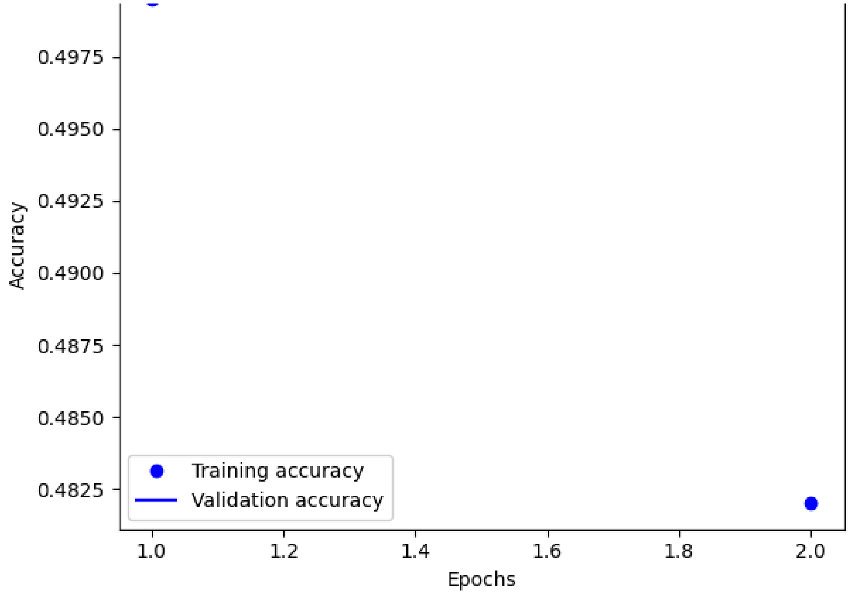

Model Base - Training and Validation Loss
Training loss
Validation loss
0.0
0.2
0.4
0.6
0.8
1.0
1.2
Loss
2
4
6
8
10
12
Epochs

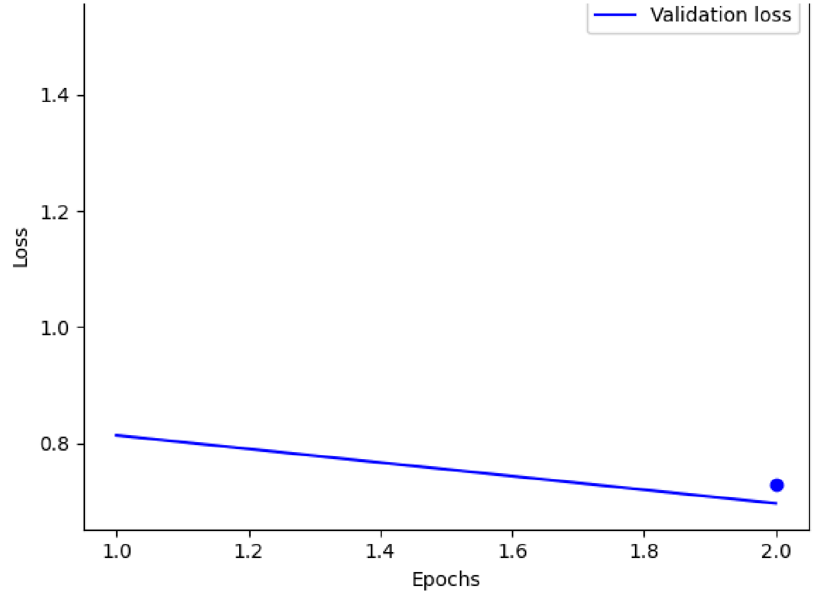

Model with Dropout - Training and Validation Accuracy
Training accuracy
Validation accuracy
0.70
0.75
0.80
0.85
0.90
0.95
Accuracy
2
4
6
8
10
12
14
16
Epochs

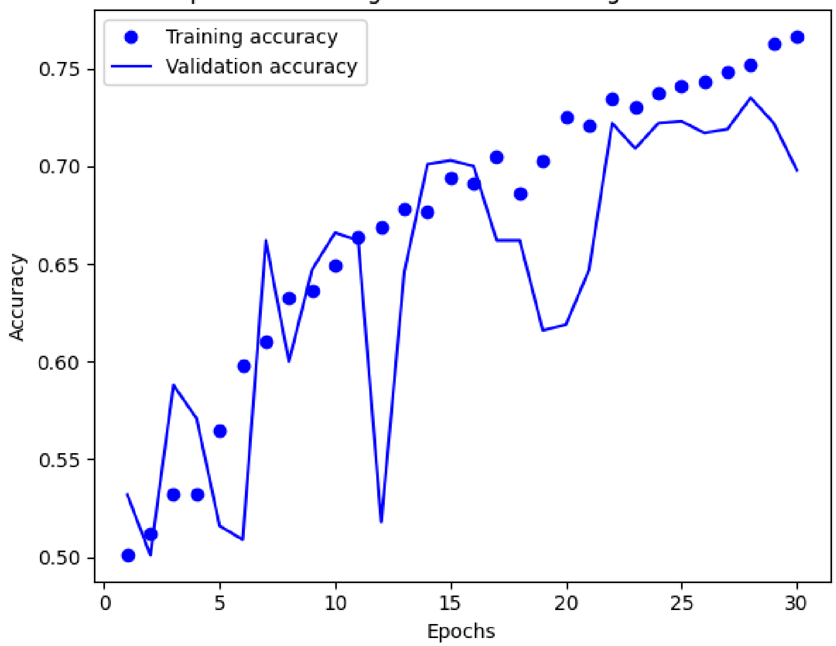

Model with Dropout - Training and Validation Loss
Training loss
Validation loss
0.2
0.4
0.6
0.8
1.0
1.2
1.4
Loss
2
4
6
8
10
12
14
16
Epochs

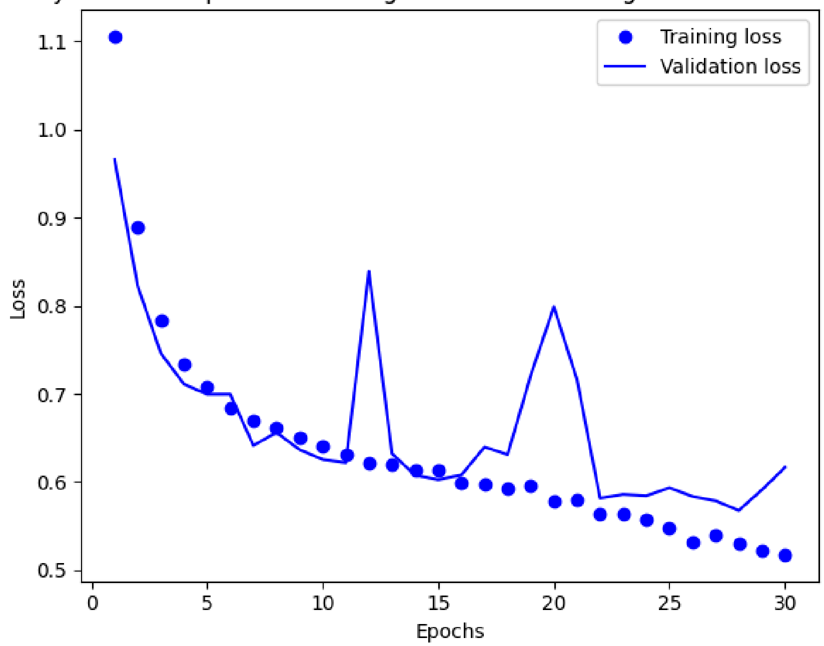

Model with L2 Regularization - Training and Validation Accuracy
0.5000

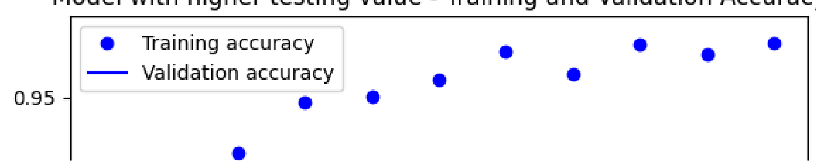

Model with L2 Regularization - Training and Validation Loss
1.6
Training loss

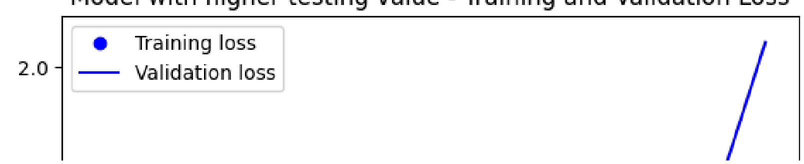

Training accuracy
Validation accuracy
Accuracy
Epochs

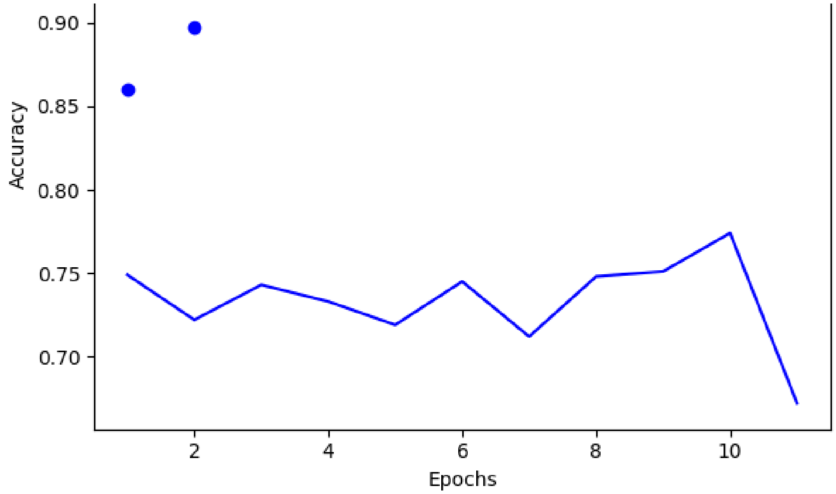

Validation loss
Loss
Epochs

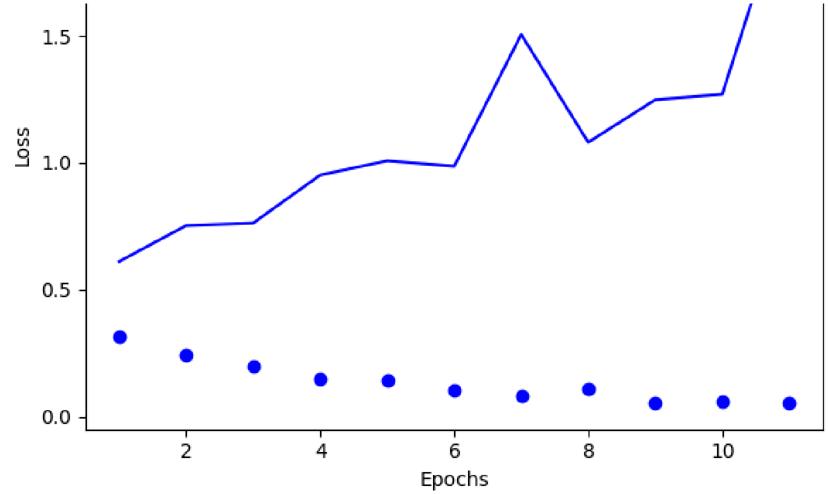

Model with Dropout and L2 Regularization - Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs

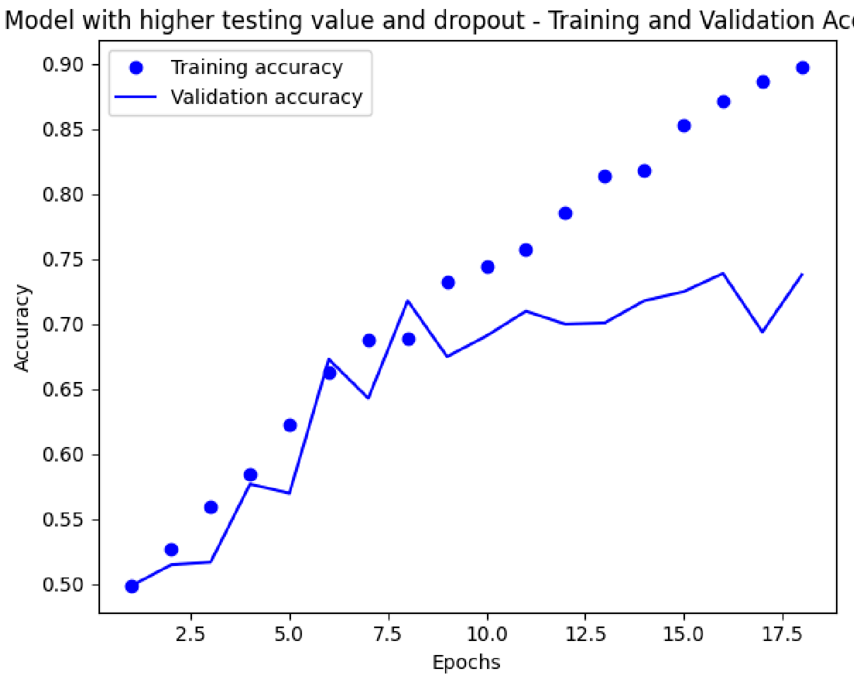

Model with Dropout and L2 Regularization - Training and Validation Loss
Training loss
Validation loss
Loss
Epochs

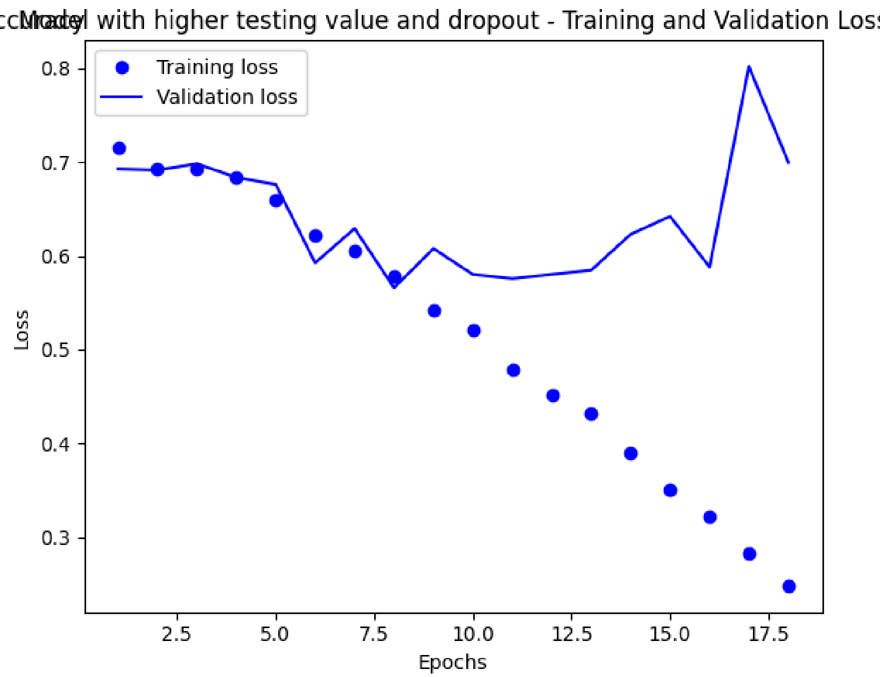

Model with higher testing value - Training and Validation Accuracy
Training accuracy
Validation accuracy


Model with higher testing value - Training and Validation Loss
Training loss
Validation loss


Accuracy
Epochs


Loss
Epochs


Model with higher testing value and dropout - Training and Validation Accuracy
Training accuracy
Validation accuracy
Accuracy
Epochs


Model with higher testing value and dropout - Training and Validation Loss
Training loss
Validation loss
Loss
Epochs

```
# Evaluate the model directly after training
test_loss, test_acc = mod_base.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 126s 4s/step - accuracy: 0.7346 - loss: 0.6816
Test accuracy: 0.718
```

```
# Evaluate the model directly after training
test_loss, test_acc = mod_d.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.7407 - loss: 0.5509
Test accuracy: 0.726
```

```
# Evaluate the model directly after training
test_loss, test_acc = mod_L2.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 107ms/step - accuracy: 0.5212 - loss: 0.6969
Test accuracy: 0.500
```

```
# Evaluate the model directly after training
test_loss, test_acc = mod_d_L2.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 100ms/step - accuracy: 0.7482 - loss: 0.5646
Test accuracy: 0.725
```

```
# Evaluate the model directly after training
test_loss, test_acc = mod_base_500.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.7354 - loss: 0.7005
Test accuracy: 0.718
```

```
# Evaluate the model directly after training
test_loss, test_acc = mod_base_1500.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 87ms/step - accuracy: 0.7249 - loss: 0.7033
Test accuracy: 0.718
```

```
# Evaluate the model directly after training
test_loss, test_acc = mod_d_2000.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 125ms/step - accuracy: 0.7500 - loss: 0.5317
Test accuracy: 0.726
```

Pretrained Model

```
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 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
conv_base.summary()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_4 (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)

```
import numpy as np

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

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 11s 11s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 202ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 196ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 184ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 201ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 189ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 191ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 187ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 178ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 190ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 199ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 186ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 198ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 193ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 184ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 180ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 188ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 184ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 175ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 179ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 174ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 182ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 163ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 185ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 176ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 177ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 171ms/step
```

```
train_features.shape
```

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

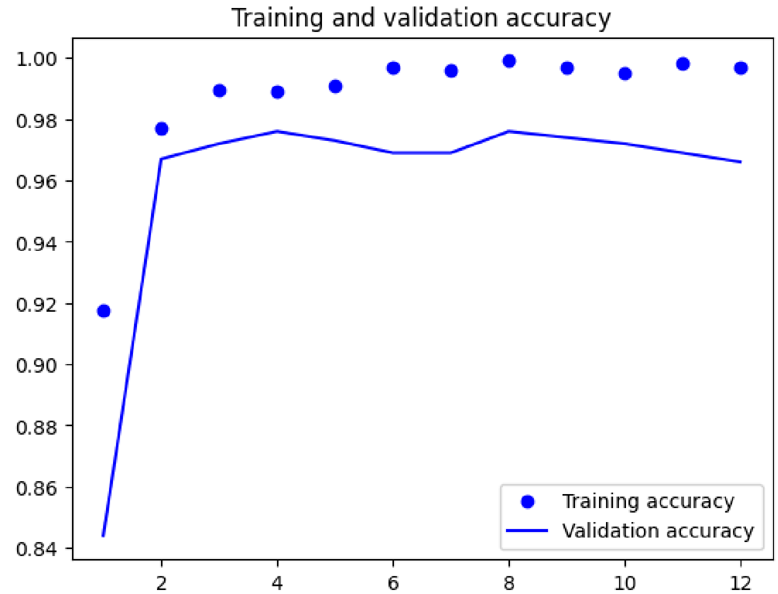
Defining and training the densely connected classifier

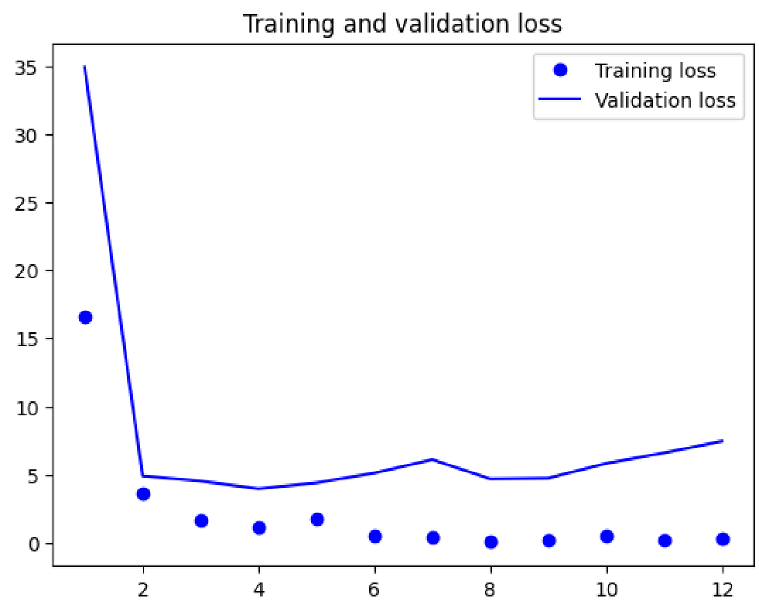
```
inputs = keras.Input(shape=(5, 5, 512))
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
          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

callbacks = [
    keras.callbacks.ModelCheckpoint(
      filepath="feature_extraction.keras",
      save_best_only=True,
      monitor="val_loss"),
    keras.callbacks.EarlyStopping(
        monitor="val_loss",
        patience=10,  # Number of epochs to wait for improvement
        restore_best_weights=True  # Restore model weights from the epoch with the best value
    )
]
history = model.fit(
    train_features, train_labels,
    epochs=12, #based on the graph highest accuracy
    validation_data=(val_features, val_labels),
    callbacks=callbacks)
```

```
Epoch 1/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 32ms/step - accuracy: 0.8639 - loss: 35.1163 - val_accuracy: 0.8440 - val_loss: 34.9084
Epoch 2/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9750 - loss: 4.7558 - val_accuracy: 0.9670 - val_loss: 4.8954
```

```
Epoch 3/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9908 - loss: 1.3448 - val_accuracy: 0.9720 - val_loss: 4.5188
Epoch 4/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9884 - loss: 0.8168 - val_accuracy: 0.9760 - val_loss: 3.9586
Epoch 5/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9902 - loss: 2.4181 - val_accuracy: 0.9730 - val_loss: 4.3880
Epoch 6/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.1311 - val_accuracy: 0.9690 - val_loss: 5.1053
Epoch 7/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9975 - loss: 0.3495 - val_accuracy: 0.9690 - val_loss: 6.0975
Epoch 8/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.0299 - val_accuracy: 0.9760 - val_loss: 4.6777
Epoch 9/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9982 - loss: 0.1206 - val_accuracy: 0.9740 - val_loss: 4.7294
Epoch 10/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9953 - loss: 0.2502 - val_accuracy: 0.9720 - val_loss: 5.8161
Epoch 11/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9995 - loss: 0.0520 - val_accuracy: 0.9690 - val_loss: 6.5862
Epoch 12/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9964 - loss: 0.4288 - val_accuracy: 0.9660 - val_loss: 7.4407
```

```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()
```






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