AML ASSIGNMENT 3

CONVOLUTION NETWORKS

AJITH RAJ PERIYASAMY

GitHub Link - https://github.com/Ajith0719/aperiyas_64061/tree/main/Assignment%203

A base model was created to see the improvements and deterioration of any changes done to the model.

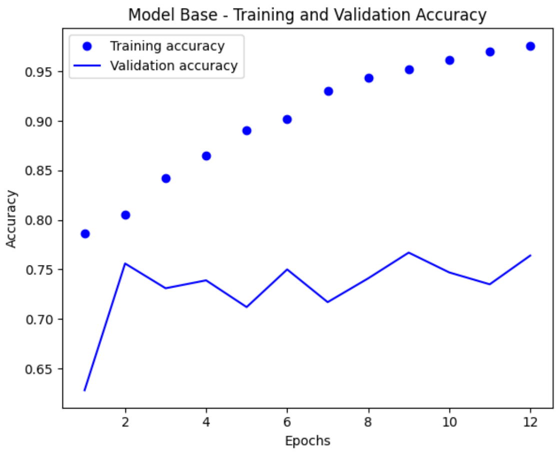
1. The below approach was taken to reduce overfitting and reduce accuracy.

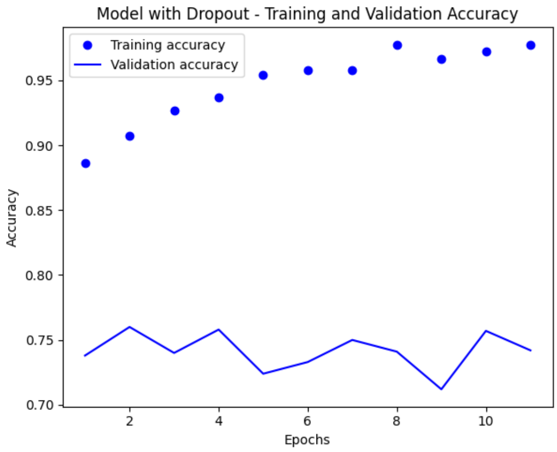
Model	**Training Image count**	**Methods**	**Training Accuracy**	**Validation Accuracy**	**Testing Accuracy**
Base	1000	none	0.95	0.76	0.72
Dropout	1000	Dropout 0.5	0.98	0.74	0.74
L2	1000	L2	0.49	0.5	0.5
Dropout & L2	1000	Dropout 0.25 & L2	0.5	0.5	0.5

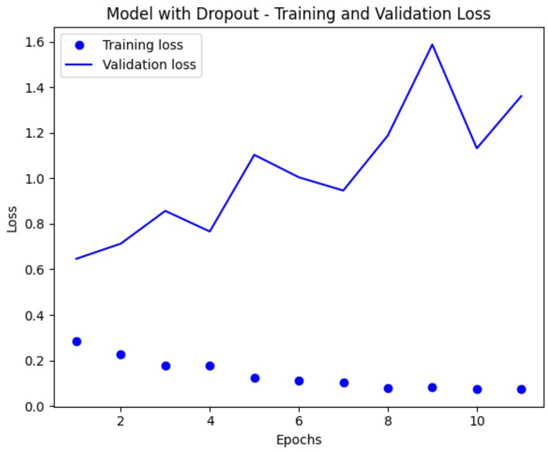
Even after incorporating dropout and L2 regularization, the model still suffered from overfitting. While the training accuracy remained high, the validation accuracy did not keep up, indicating that the model struggled to generalize effectively to new data.

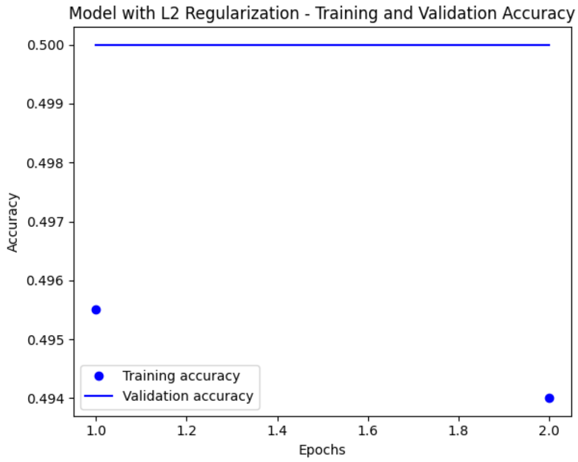
Although dropout and L2 regularization contributed to mitigating overfitting to some degree, they were not sufficient to attain high test accuracy on this relatively small dataset. This implies that either a more sophisticated model or a larger dataset would be necessary for improved performance.

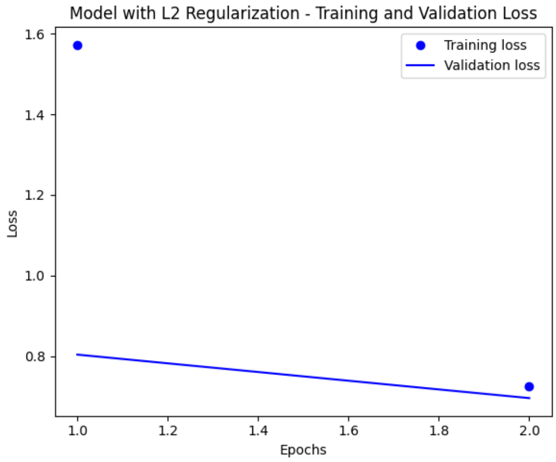
The optimal results were observed with a dropout rate of 0.5, where accuracy was the highest among all tested models. The following graphs illustrate the model performance with key metrics.

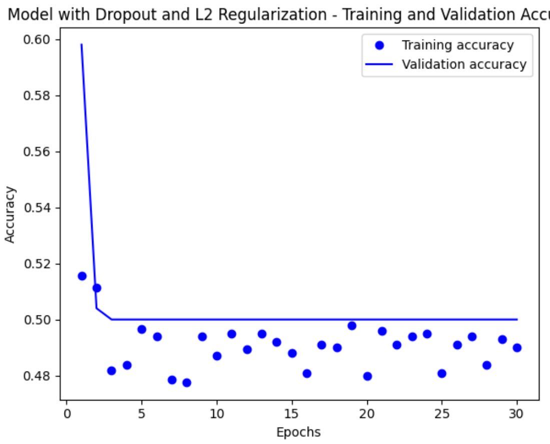


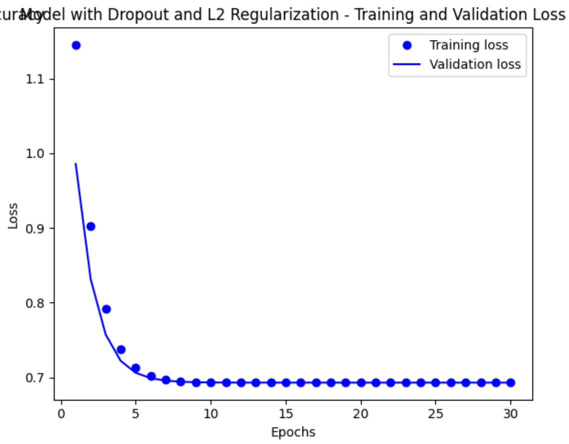






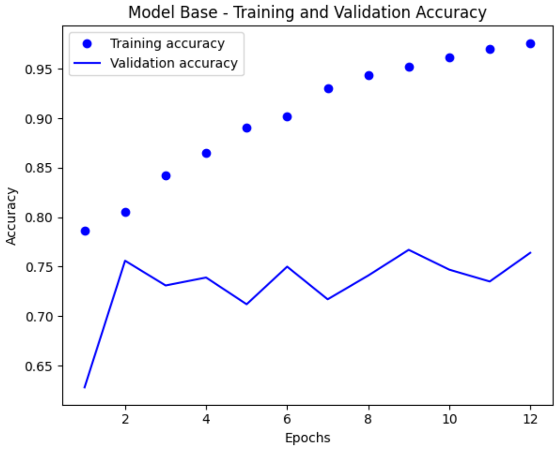


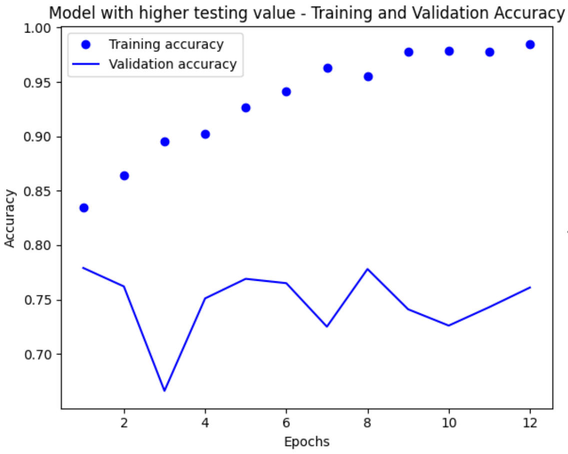


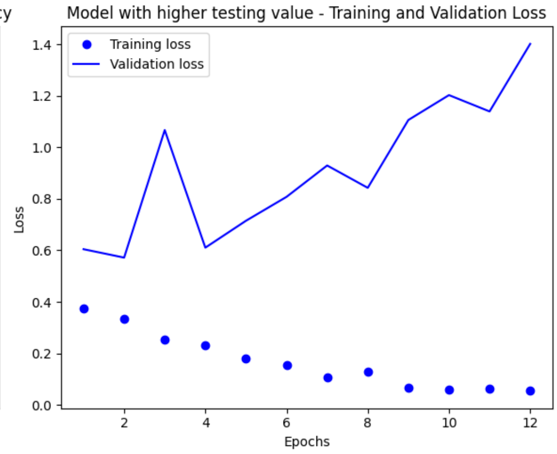


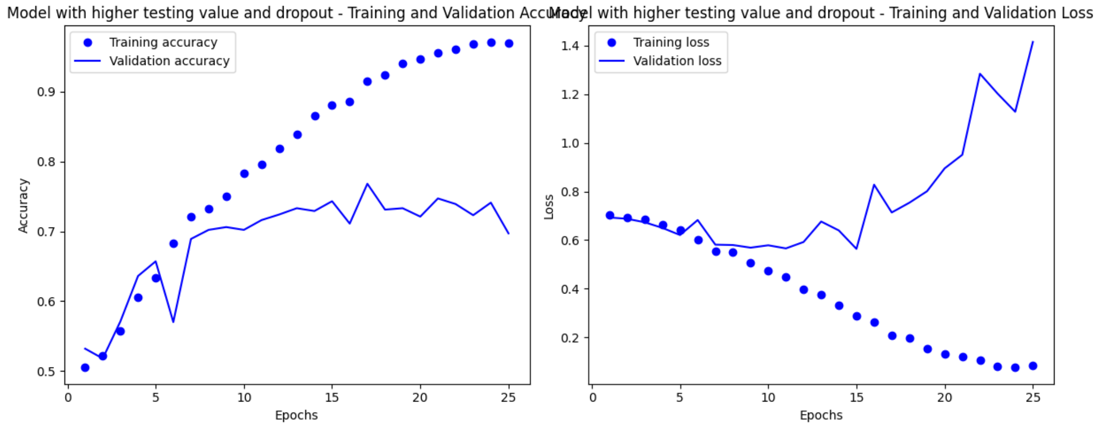
2&3. Since these questions are regrading training set size both the question was merged in one. The training set was increased by 50% (1000 to 1500) but the model.

Model	Training Image count	Methods	Training Accuracy	Validation Accuracy	Testing Accuracy
Base	1000	none	0.95	0.76	0.72
Model 1500	1500	none	0.98	0.76	0.73
Dropout and 2000	2000	Dropout	0.98	0.70	0.73









2. Model_base_1500:

(The same base model was utilized, but with 1500 images for training.)

- **New Training Sample Size:** 1500
- **Validation and Test Sample Sizes:** 500 each (unchanged from Step 1).
- **Model Configuration:** A similar CNN architecture was trained from scratch, incorporating dropout and L2 regularization, just as in Step 1.
- **Objective:** Assess whether increasing the training dataset size enhances model performance when training from scratch.

Results:

- **Accuracy:** 0.73

Observations:

- **Minimal improvement:** The test accuracy remained at 0.73, showing only a slight difference compared to the model trained with 1000 samples. This suggests that the additional 500 samples did not introduce enough variety to significantly enhance the model's generalization capability.
- **Potential Reasons:** The limited improvement could stem from the model's relatively simple architecture, which may have already reached its capacity to extract meaningful features given the dataset size. Alternatively, the additional images may have been too like the existing ones, contributing little in terms of feature diversity.

3. Model_d_2000:

(This model incorporated a dropout rate of 0.25 and was trained with 2000 images.)

- **Adjusted Training Sample Size:** 2000

- **Validation and Test Sample Sizes:** 500 each
- **Model Configuration:** The same CNN architecture was trained from scratch, utilizing dropout and L2 regularization.
- **Objective:** Evaluate whether increasing the training sample size enhances generalization and accuracy.

Results:

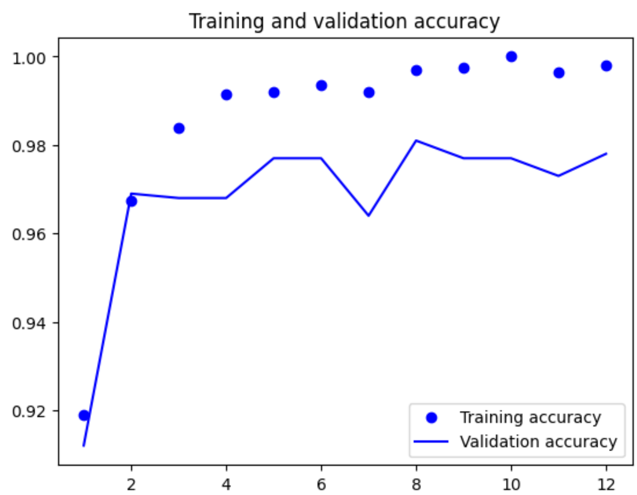
- **Test Accuracy:** 0.73

Observations:

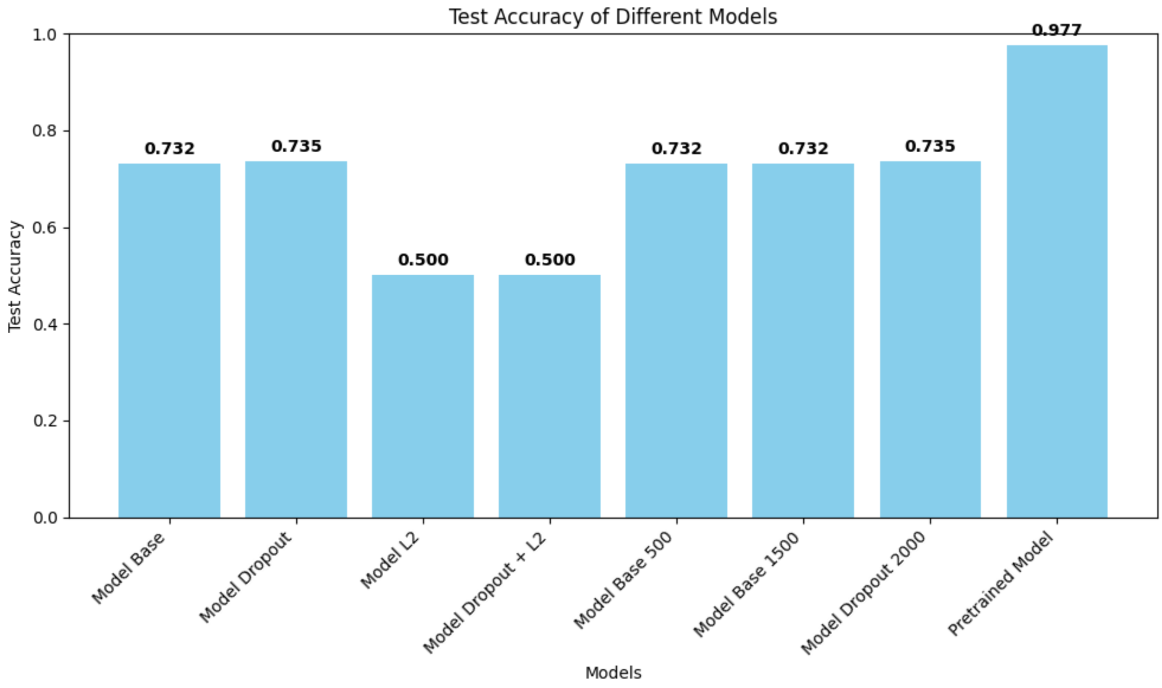
- **Performance Gains:** Expanding the training dataset to 2000 images resulted in only a marginal increase in test accuracy, indicating that the model did not significantly benefit from the additional data.
- **Continued Overfitting:** Although there was a slight improvement in accuracy, the model still exhibited signs of overfitting. The training accuracy consistently exceeded the validation accuracy, suggesting that additional regularization or a more complex model might be necessary to enhance performance.
- **Comparison Insight:** Among the models trained from scratch, using 2000 training samples yielded the highest performance. However, further improvements may require either a more advanced model architecture or an even larger dataset.

4. Optimizing pre-trained model:

In this section, we pre-trained a VGG16 model on ImageNet and fine-tuned it on the Cats & Dogs dataset using the same sample sizes as the from-scratch training.



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	0.98	0.74	0.74
L2	1000	L2	0.49	0.5	0.5
Dropout & L2	1000	Dropout 0.25 & L2	0.5	0.5	0.5
Model 1500	1500	none	0.98	0.76	0.72
Dropout and 2000	2000	Dropout	0.98	0.70	0.73
Pretrained	NA	Dropout	0.96	0.97	0.977



Results: Accuracy: 0.977

Observations:

The pre-trained model has the highest accuracy (0.977) with 2000 training samples. This finding implies that, while pre-trained models perform well with minimal data, more samples can improve their accuracy, especially in finetuning tasks. Minimal Overfitting: The pre-trained model exhibited minimal overfitting, with training and validation accuracy being closely aligned. This demonstrates the durability of pre-trained models in achieving high accuracy while minimizing overfitting.

Training Sample Size and Model Performance:

For models trained from scratch, increasing the sample size from 1000 to 2000 resulted in a slight increase in accuracy (0.732 to 0.735). However, even with higher sample sets, models built from scratch performed worse than pre-trained models.

Pretrained Models vs. From Scratch Models: The pre-trained VGG16 model consistently outperformed the from-scratch models, with a high accuracy of 0.977 after only 1000 training samples. This highlights the benefit of transfer learning for short datasets since pre-trained models can use existing feature representations to achieve excellent performance with limited data.

Recommendations:

For datasets of small to moderate size, utilizing pre-trained models offers a clear performance advantage and is the preferred approach. When working with larger datasets, training a model from scratch can be viable, especially when combined with strong regularization techniques and appropriate model adjustments.

This study highlights the effectiveness of transfer learning while also demonstrating the challenges of training from scratch with limited data. These insights can serve as a valuable guide for selecting models in similar classification tasks.

Base Model [Code provided by Professor edit done - Training 1000 Validation 500 and Testing 500, 1500 images in total including cats and dogs)

```
import os
import shutil
import pathlib

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

image_path = pathlib.Path("/content/drive/MyDrive/Colab Notebooks/cats_vs_dogs_small")
```

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

```
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)

model_base = keras.Model(inputs=x_base_input, outputs=x_base_output)
model_base_1500 = keras.Model(inputs=x_base_input, outputs=x_base_output)
model_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)

model_d = keras.Model(inputs=x_d_input, outputs=x_d_output)
model_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_res
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_poc
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_pc
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_pc
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_pc
x_L2_flatten = layers.Flatten()(x_L2_conv5)
x_L2_output = layers.Dense(1, activation="sigmoid")(x_L2_flatten)

model_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
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
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_L
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_L
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_L
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)

model_d_L2 = keras.Model(inputs=x_d_L2_input, outputs=x_d_L2_output)
```

```
model_base.summary()
```

Show hidden output

```
model_d.summary()
```

Show hidden output

```
model_L2.summary()
```

Show hidden output

```
model_d_L2.summary()
```

Show hidden output

Configuring all models for training

Compiling all the Keras models to tensorflow - SMD

Double-click (or enter) to edit

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

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

```
model_L2.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
model_d_L2.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
model_base_1500.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
model_d_2000.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
model_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 = 42
tf.random.set_seed(seed)

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

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

test_full_dataset = image_dataset_from_directory(
    image_path / "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,)
```

Fitting the model to dataset

```
# 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
history_base = model_base.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 492s 8s/step - accuracy: 0.5110 - loss: 0.7277 - val_accuracy: 0.5000 - val_loss: 0.6925
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 139ms/step - accuracy: 0.5101 - loss: 0.6928 - val_accuracy: 0.5000 - val_loss: 0.7261
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.5597 - loss: 0.6908 - val_accuracy: 0.6220 - val_loss: 0.6629
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.6267 - loss: 0.6567 - val_accuracy: 0.6350 - val_loss: 0.6249
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.6677 - loss: 0.6333 - val_accuracy: 0.6790 - val_loss: 0.6000
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.6705 - loss: 0.6172 - val_accuracy: 0.7020 - val_loss: 0.5778
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.7169 - loss: 0.5623 - val_accuracy: 0.6570 - val_loss: 0.6752
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 181ms/step - accuracy: 0.7192 - loss: 0.5337 - val_accuracy: 0.6990 - val_loss: 0.5694
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.7597 - loss: 0.4909 - val_accuracy: 0.7070 - val_loss: 0.5548
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 149ms/step - accuracy: 0.7903 - loss: 0.4543 - val_accuracy: 0.7200 - val_loss: 0.6650
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.8071 - loss: 0.4155 - val_accuracy: 0.7470 - val_loss: 0.5649
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.8432 - loss: 0.3638 - val_accuracy: 0.7330 - val_loss: 0.6410
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 136ms/step - accuracy: 0.8696 - loss: 0.3242 - val_accuracy: 0.7120 - val_loss: 0.8624
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 178ms/step - accuracy: 0.8788 - loss: 0.2939 - val_accuracy: 0.6990 - val_loss: 0.9355
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 150ms/step - accuracy: 0.9023 - loss: 0.2639 - val_accuracy: 0.7260 - val_loss: 0.8734
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.9273 - loss: 0.1772 - val_accuracy: 0.7520 - val_loss: 0.8081
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 181ms/step - accuracy: 0.9421 - loss: 0.1632 - val_accuracy: 0.7230 - val_loss: 0.9632
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 19s 157ms/step - accuracy: 0.9447 - loss: 0.1456 - val_accuracy: 0.7470 - val_loss: 1.0273
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9521 - loss: 0.1237 - val_accuracy: 0.7650 - val_loss: 0.8937
```

```
# Fit the base model
history_base = model_base_500.fit(
    train_dataset_500,
    epochs=30,
    validation_data=validation_dataset,
```

```
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 174ms/step - accuracy: 0.7617 - loss: 0.4908 - val_accuracy: 0.6280 - val_loss: 0.7463
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.7844 - loss: 0.4457 - val_accuracy: 0.7560 - val_loss: 0.5527
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 189ms/step - accuracy: 0.8204 - loss: 0.3815 - val_accuracy: 0.7310 - val_loss: 0.6283
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.8624 - loss: 0.3196 - val_accuracy: 0.7390 - val_loss: 0.6279
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.8828 - loss: 0.2669 - val_accuracy: 0.7120 - val_loss: 0.9690
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9043 - loss: 0.2388 - val_accuracy: 0.7500 - val_loss: 0.7939
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.9237 - loss: 0.1772 - val_accuracy: 0.7170 - val_loss: 0.8999
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 144ms/step - accuracy: 0.9363 - loss: 0.1408 - val_accuracy: 0.7410 - val_loss: 1.1190
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 164ms/step - accuracy: 0.9468 - loss: 0.1362 - val_accuracy: 0.7670 - val_loss: 0.9175
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.9626 - loss: 0.0942 - val_accuracy: 0.7470 - val_loss: 1.0731
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.9734 - loss: 0.0687 - val_accuracy: 0.7350 - val_loss: 1.1853
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 140ms/step - accuracy: 0.9744 - loss: 0.0776 - val_accuracy: 0.7640 - val_loss: 1.2436
```

```
# Fit the base model
history_base_1500 = model_base_1500.fit(
    train_dataset_1500,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 161ms/step - accuracy: 0.8135 - loss: 0.4243 - val_accuracy: 0.7790 - val_loss: 0.6046
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.8568 - loss: 0.3554 - val_accuracy: 0.7620 - val_loss: 0.5719
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 190ms/step - accuracy: 0.9014 - loss: 0.2491 - val_accuracy: 0.6660 - val_loss: 1.0671
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.9013 - loss: 0.2411 - val_accuracy: 0.7510 - val_loss: 0.6106
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 138ms/step - accuracy: 0.9390 - loss: 0.1713 - val_accuracy: 0.7690 - val_loss: 0.7149
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 140ms/step - accuracy: 0.9505 - loss: 0.1243 - val_accuracy: 0.7650 - val_loss: 0.8078
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9624 - loss: 0.0977 - val_accuracy: 0.7250 - val_loss: 0.9295
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9721 - loss: 0.0930 - val_accuracy: 0.7780 - val_loss: 0.8427
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 8s 134ms/step - accuracy: 0.9843 - loss: 0.0561 - val_accuracy: 0.7410 - val_loss: 1.1061
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9800 - loss: 0.0705 - val_accuracy: 0.7260 - val_loss: 1.2027
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.9768 - loss: 0.0733 - val_accuracy: 0.7430 - val_loss: 1.1393
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 155ms/step - accuracy: 0.9817 - loss: 0.0636 - val_accuracy: 0.7610 - val_loss: 1.4016
```

```
# Fit the base model
history_d_2000 = model_d_2000.fit(
    train_dataset_2000,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 18s 144ms/step - accuracy: 0.4911 - loss: 0.7223 - val_accuracy: 0.5320 - val_loss: 0.6928
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 115ms/step - accuracy: 0.5168 - loss: 0.6944 - val_accuracy: 0.5180 - val_loss: 0.6873
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 17s 82ms/step - accuracy: 0.5502 - loss: 0.6914 - val_accuracy: 0.5710 - val_loss: 0.6730
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 115ms/step - accuracy: 0.5943 - loss: 0.6720 - val_accuracy: 0.6360 - val_loss: 0.6499
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 78ms/step - accuracy: 0.6181 - loss: 0.6496 - val_accuracy: 0.6570 - val_loss: 0.6205
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 76ms/step - accuracy: 0.6746 - loss: 0.6040 - val_accuracy: 0.5700 - val_loss: 0.6822
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 77ms/step - accuracy: 0.7022 - loss: 0.5710 - val_accuracy: 0.6890 - val_loss: 0.5809
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 91ms/step - accuracy: 0.7438 - loss: 0.5349 - val_accuracy: 0.7020 - val_loss: 0.5794
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 87ms/step - accuracy: 0.7496 - loss: 0.5018 - val_accuracy: 0.7060 - val_loss: 0.5685
```

```
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 77ms/step - accuracy: 0.7842 - loss: 0.4639 - val_accuracy: 0.7020 - val_loss: 0.5783
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 78ms/step - accuracy: 0.8059 - loss: 0.4419 - val_accuracy: 0.7160 - val_loss: 0.5655
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 115ms/step - accuracy: 0.8229 - loss: 0.3793 - val_accuracy: 0.7240 - val_loss: 0.5918
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 115ms/step - accuracy: 0.8346 - loss: 0.3801 - val_accuracy: 0.7330 - val_loss: 0.6761
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 81ms/step - accuracy: 0.8697 - loss: 0.3190 - val_accuracy: 0.7290 - val_loss: 0.6394
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 114ms/step - accuracy: 0.8821 - loss: 0.2826 - val_accuracy: 0.7430 - val_loss: 0.5635
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 90ms/step - accuracy: 0.8902 - loss: 0.2585 - val_accuracy: 0.7110 - val_loss: 0.8278
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 23s 114ms/step - accuracy: 0.9141 - loss: 0.1977 - val_accuracy: 0.7680 - val_loss: 0.7134
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 115ms/step - accuracy: 0.9285 - loss: 0.1798 - val_accuracy: 0.7310 - val_loss: 0.7536
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 115ms/step - accuracy: 0.9498 - loss: 0.1353 - val_accuracy: 0.7330 - val_loss: 0.8008
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 76ms/step - accuracy: 0.9471 - loss: 0.1246 - val_accuracy: 0.7210 - val_loss: 0.8951
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 77ms/step - accuracy: 0.9627 - loss: 0.1001 - val_accuracy: 0.7470 - val_loss: 0.9510
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 81ms/step - accuracy: 0.9667 - loss: 0.0974 - val_accuracy: 0.7390 - val_loss: 1.2846
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 114ms/step - accuracy: 0.9689 - loss: 0.0756 - val_accuracy: 0.7230 - val_loss: 1.2028
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 114ms/step - accuracy: 0.9599 - loss: 0.0987 - val_accuracy: 0.7410 - val_loss: 1.1280
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 17s 79ms/step - accuracy: 0.9755 - loss: 0.0714 - val_accuracy: 0.6970 - val_loss: 1.4156
```

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

```
Epoch 1/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 176ms/step - accuracy: 0.8717 - loss: 0.3399 - val_accuracy: 0.7380 - val_loss: 0.6463
Epoch 2/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 149ms/step - accuracy: 0.8954 - loss: 0.2479 - val_accuracy: 0.7600 - val_loss: 0.7130
Epoch 3/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 134ms/step - accuracy: 0.9206 - loss: 0.1864 - val_accuracy: 0.7400 - val_loss: 0.8568
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 150ms/step - accuracy: 0.9302 - loss: 0.1776 - val_accuracy: 0.7580 - val_loss: 0.7662
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.9576 - loss: 0.1126 - val_accuracy: 0.7240 - val_loss: 1.1028
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 145ms/step - accuracy: 0.9562 - loss: 0.1121 - val_accuracy: 0.7330 - val_loss: 1.0044
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 144ms/step - accuracy: 0.9642 - loss: 0.0942 - val_accuracy: 0.7500 - val_loss: 0.9461
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 194ms/step - accuracy: 0.9803 - loss: 0.0705 - val_accuracy: 0.7410 - val_loss: 1.1876
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 193ms/step - accuracy: 0.9621 - loss: 0.0859 - val_accuracy: 0.7120 - val_loss: 1.5871
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 192ms/step - accuracy: 0.9745 - loss: 0.0765 - val_accuracy: 0.7570 - val_loss: 1.1316
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 191ms/step - accuracy: 0.9799 - loss: 0.0591 - val_accuracy: 0.7420 - val_loss: 1.3611
```

```
history_L2 = model_L2.fit(
    train_dataset,
    epochs=2, # Best so stopping at 2
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 1/2
63/63 ━━━━━━━━━━━━━━━━━━━━ 16s 201ms/step - accuracy: 0.5015 - loss: 2.2442 - val_accuracy: 0.5000 - val_loss: 0.8042
Epoch 2/2
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 171ms/step - accuracy: 0.4934 - loss: 0.7521 - val_accuracy: 0.5000 - val_loss: 0.6964
```

```
history_d_L2 = model_d_L2.fit(
    train_dataset,
    epochs=30,
    validation_data=validation_dataset,
    callbacks=callbacks
)
```

```
Epoch 2/30
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 199ms/step - accuracy: 0.5006 - loss: 0.8100 - val_accuracy: 0.5000 - val_loss: 0.7575
Epoch 4/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 162ms/step - accuracy: 0.4791 - loss: 0.7470 - val_accuracy: 0.5000 - val_loss: 0.7223
Epoch 5/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 181ms/step - accuracy: 0.5013 - loss: 0.7176 - val_accuracy: 0.5000 - val_loss: 0.7063
Epoch 6/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 142ms/step - accuracy: 0.5019 - loss: 0.7043 - val_accuracy: 0.5000 - val_loss: 0.6991
Epoch 7/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.4789 - loss: 0.6983 - val_accuracy: 0.5000 - val_loss: 0.6958
Epoch 8/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 195ms/step - accuracy: 0.4868 - loss: 0.6955 - val_accuracy: 0.5000 - val_loss: 0.6944
Epoch 9/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.4940 - loss: 0.6943 - val_accuracy: 0.5000 - val_loss: 0.6937
Epoch 10/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 139ms/step - accuracy: 0.4817 - loss: 0.6937 - val_accuracy: 0.5000 - val_loss: 0.6934
Epoch 11/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 147ms/step - accuracy: 0.5002 - loss: 0.6934 - val_accuracy: 0.5000 - val_loss: 0.6933
Epoch 12/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.4900 - loss: 0.6933 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 13/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 156ms/step - accuracy: 0.4971 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 14/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 136ms/step - accuracy: 0.4977 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 15/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 154ms/step - accuracy: 0.4937 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 16/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.4892 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 17/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 141ms/step - accuracy: 0.4934 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6932
Epoch 18/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 174ms/step - accuracy: 0.4865 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 19/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 144ms/step - accuracy: 0.5008 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 20/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 153ms/step - accuracy: 0.4741 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 21/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 175ms/step - accuracy: 0.4959 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 22/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 9s 140ms/step - accuracy: 0.4884 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 23/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 159ms/step - accuracy: 0.5032 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 24/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 160ms/step - accuracy: 0.4974 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 25/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 172ms/step - accuracy: 0.4783 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 26/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 22s 188ms/step - accuracy: 0.4964 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 27/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 21s 196ms/step - accuracy: 0.4977 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 28/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 157ms/step - accuracy: 0.4830 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 29/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 161ms/step - accuracy: 0.4898 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
Epoch 30/30
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.5086 - loss: 0.6932 - val_accuracy: 0.5000 - val_loss: 0.6931
```

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(history_base, "Model Base")
plot_training_history(history_d, "Model with Dropout")
plot_training_history(history_L2, "Model with L2 Regularization")
plot_training_history(history_d_L2, "Model with Dropout and L2 Regularization")
plot_training_history(history_base_1500, "Model with higher testing value")
plot_training_history(history_d_2000, "Model with higher testing value and dropout")
```

Show hidden output

Checking accuracy in test set

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 231s 7s/step - accuracy: 0.7237 - loss: 0.6511
Test accuracy: 0.732
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.7422 - loss: 0.6445
Test accuracy: 0.735
```

```
# Evaluate the model directly after training
test_loss, test_acc = model_L2.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 87ms/step - accuracy: 0.5192 - loss: 0.6959
Test accuracy: 0.500
```

```
# Evaluate the model directly after training
test_loss, test_acc = model_d_L2.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 118ms/step - accuracy: 0.5099 - loss: 0.6931
Test accuracy: 0.500
```

```
# Evaluate the model directly after training
test_loss, test_acc = model_base_500.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.7295 - loss: 0.6139
Test accuracy: 0.732
```

```
# Evaluate the model directly after training
test_loss, test_acc = model_base_1500.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 86ms/step - accuracy: 0.7283 - loss: 0.6411
Test accuracy: 0.732
```

```
# Evaluate the model directly after training
test_loss, test_acc = model_d_2000.evaluate(test_dataset)
print(f"Test accuracy: {test_acc:.3f}")
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 92ms/step - accuracy: 0.7391 - loss: 0.6409
Test accuracy: 0.735
```

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_t
58889256/58889256 ━━━━━━━━━━━━━━━━━━━━ 2s 0us/step
```

```
conv_base.summary()
```

Show hidden output

```
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)
```

Show hidden output

```
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 ━━━━━━━━━━━━━━━━━━━━ 4s 44ms/step - accuracy: 0.8410 - loss: 49.3645 - val_accuracy: 0.9120 - val_loss: 14.7128
Epoch 2/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 15ms/step - accuracy: 0.9647 - loss: 4.8894 - val_accuracy: 0.9690 - val_loss: 4.9459
Epoch 3/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9861 - loss: 1.5468 - val_accuracy: 0.9680 - val_loss: 5.8455
Epoch 4/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9941 - loss: 0.5080 - val_accuracy: 0.9680 - val_loss: 6.0097
Epoch 5/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9937 - loss: 0.8348 - val_accuracy: 0.9770 - val_loss: 4.6024
Epoch 6/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9950 - loss: 0.8701 - val_accuracy: 0.9770 - val_loss: 4.8669
Epoch 7/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9945 - loss: 0.3545 - val_accuracy: 0.9640 - val_loss: 8.6939
Epoch 8/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - accuracy: 0.9928 - loss: 1.3516 - val_accuracy: 0.9810 - val_loss: 3.5532
Epoch 9/12
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9984 - loss: 0.3088 - val_accuracy: 0.9770 - val_loss: 4.0285
Epoch 10/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 1.0000 - loss: 1.0011e-26 - val_accuracy: 0.9770 - val_loss: 4.0285
Epoch 11/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9966 - loss: 0.1482 - val_accuracy: 0.9730 - val_loss: 4.3043
Epoch 12/12
63/63 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.0746 - val_accuracy: 0.9780 - val_loss: 3.7556
```

```
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()
```

Show hidden output

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

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

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

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

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

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

inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = keras.applications.vgg16.preprocess_input(x)
x = conv_base(x)
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs, outputs)
model.compile(loss="binary_crossentropy",
              optimizer="rmsprop",
              metrics=["accuracy"])
```

```
conv_base.summary()
```

Show hidden output

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

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

callbacks = [
```

```
    keras.callbacks.ModelCheckpoint(
        filepath="fine_tuning.keras",
        save_best_only=True,
        monitor="val_loss"),
    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_dataset,
    epochs=11, # best accuracy
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 20s 219ms/step - accuracy: 0.6693 - loss: 5.7583 - val_accuracy: 0.9270 - val_loss: 0.5773
Epoch 2/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 14s 181ms/step - accuracy: 0.8307 - loss: 1.5033 - val_accuracy: 0.9530 - val_loss: 0.3627
Epoch 3/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 202ms/step - accuracy: 0.8955 - loss: 0.6786 - val_accuracy: 0.9620 - val_loss: 0.2878
Epoch 4/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 197ms/step - accuracy: 0.9250 - loss: 0.3847 - val_accuracy: 0.9600 - val_loss: 0.2386
Epoch 5/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 13s 212ms/step - accuracy: 0.9283 - loss: 0.3065 - val_accuracy: 0.9720 - val_loss: 0.1866
Epoch 6/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.9443 - loss: 0.2567 - val_accuracy: 0.9680 - val_loss: 0.1689
Epoch 7/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 10s 162ms/step - accuracy: 0.9511 - loss: 0.1708 - val_accuracy: 0.9650 - val_loss: 0.1875
Epoch 8/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 179ms/step - accuracy: 0.9434 - loss: 0.2178 - val_accuracy: 0.9630 - val_loss: 0.1648
Epoch 9/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 176ms/step - accuracy: 0.9564 - loss: 0.1490 - val_accuracy: 0.9700 - val_loss: 0.1705
Epoch 10/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 12s 183ms/step - accuracy: 0.9716 - loss: 0.1175 - val_accuracy: 0.9720 - val_loss: 0.1555
Epoch 11/11
63/63 ━━━━━━━━━━━━━━━━━━━━ 11s 175ms/step - accuracy: 0.9639 - loss: 0.1450 - val_accuracy: 0.9720 - val_loss: 0.1864
```

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

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 99ms/step - accuracy: 0.9836 - loss: 0.1264
Test accuracy: 0.977
```

```
# Evaluate each model and store the accuracy
model_base_test_loss, model_base_test_acc = model_base.evaluate(test_dataset)
model_d_test_loss, model_d_test_acc = model_d.evaluate(test_dataset)
model_L2_test_loss, model_L2_test_acc = model_L2.evaluate(test_dataset)
model_d_L2_test_loss, model_d_L2_test_acc = model_d_L2.evaluate(test_dataset)
model_base_500_test_loss, model_base_500_test_acc = model_base_500.evaluate(test_dataset)
model_base_1500_test_loss, model_base_1500_test_acc = model_base_1500.evaluate(test_dataset)
model_d_2000_test_loss, model_d_2000_test_acc = model_d_2000.evaluate(test_dataset)
pretrained_model_test_loss, pretrained_model_test_acc = model.evaluate(test_dataset)

# Now, let's put these accuracies in a list for plotting
test_accuracies = [
    model_base_test_acc, model_d_test_acc, model_L2_test_acc, model_d_L2_test_acc,
    model_base_500_test_acc, model_base_1500_test_acc, model_d_2000_test_acc, pretrained_model_test_acc
]

# Names of the models for display on the plot
model_names = [
    "Model Base", "Model Dropout", "Model L2", "Model Dropout + L2",
    "Model Base 500", "Model Base 1500", "Model Dropout 2000", "Pretrained Model"
]

# Plotting
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.bar(model_names, test_accuracies, color='skyblue')
plt.xlabel("Models")
plt.ylabel("Test Accuracy")
plt.title("Test Accuracy of Different Models")
plt.xticks(rotation=45, ha="right")
plt.ylim(0, 1)  # Assuming accuracy is between 0 and 1

# Show the accuracy values on top of each bar
for i, v in enumerate(test_accuracies):
    plt.text(i, v + 0.02, f"{v:.3f}", ha="center", fontweight="bold")
```

```
plt.tight_layout()
plt.show()
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 88ms/step - accuracy: 0.7358 - loss: 0.5965
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 120ms/step - accuracy: 0.7514 - loss: 0.6130
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 92ms/step - accuracy: 0.4912 - loss: 0.6966
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 90ms/step - accuracy: 0.4999 - loss: 0.6931
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 91ms/step - accuracy: 0.7180 - loss: 0.6611
32/32 ━━━━━━━━━━━━━━━━━━━━ 4s 111ms/step - accuracy: 0.7428 - loss: 0.6028
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.7125 - loss: 0.6775
32/32 ━━━━━━━━━━━━━━━━━━━━ 3s 96ms/step - accuracy: 0.9791 - loss: 0.1471
```