

Assignment 3 – Convolution

**Advanced Machine Learning (BA-64061-001)**

vvedula@kent.edu

GitHub: [Assignment 3 GitHub](#)

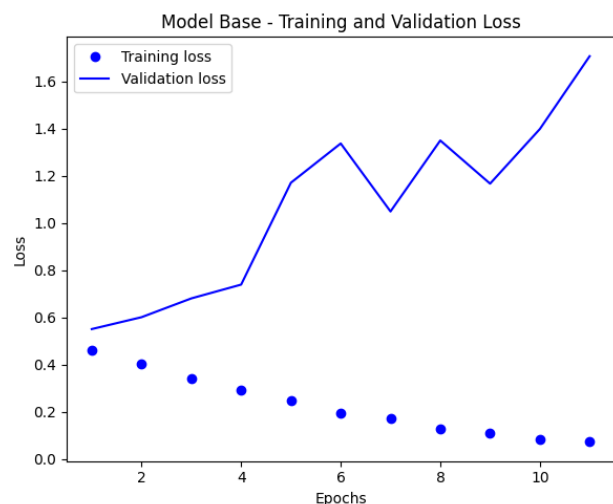
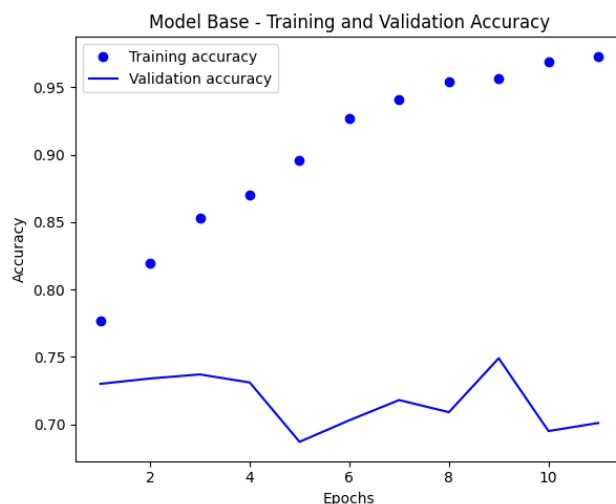
## About the assignment :

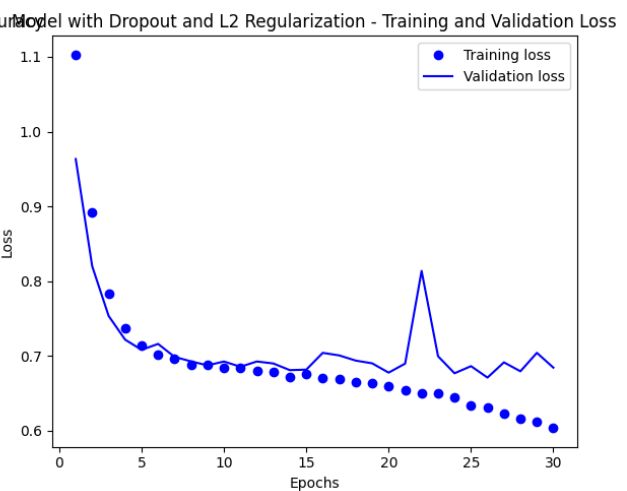
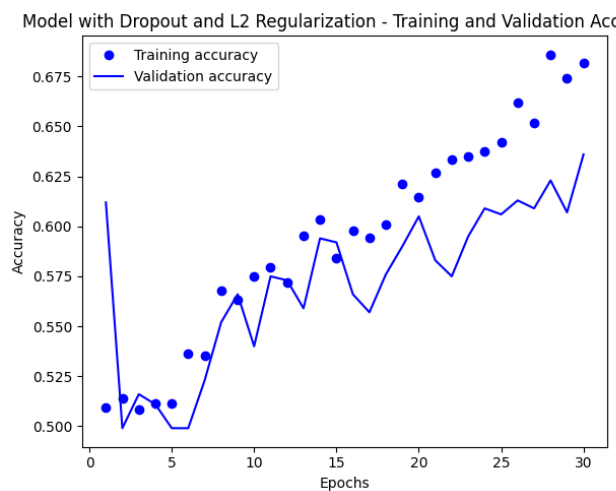
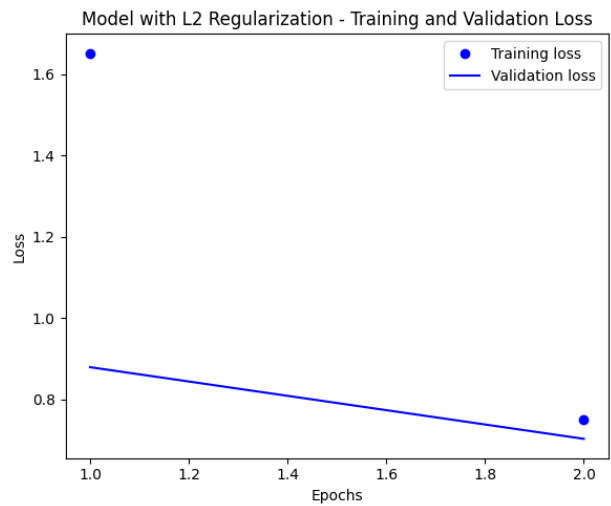
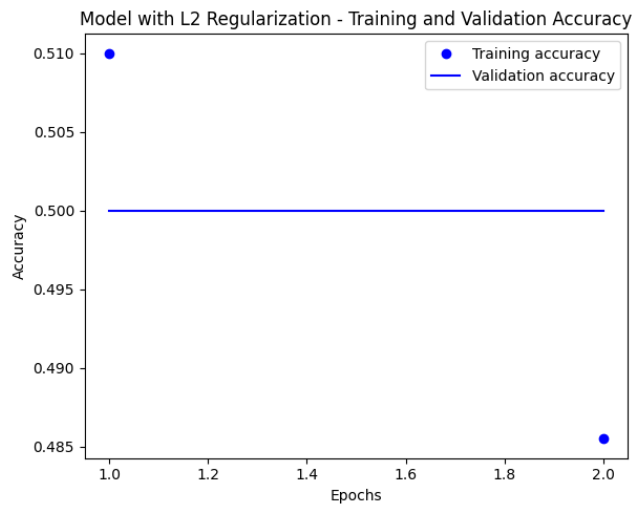
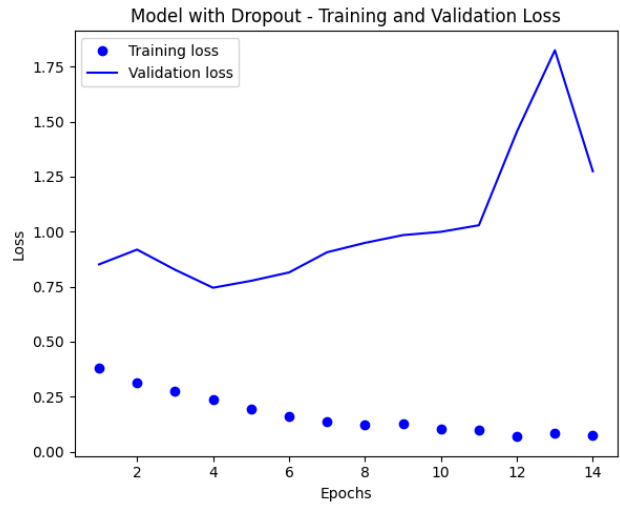
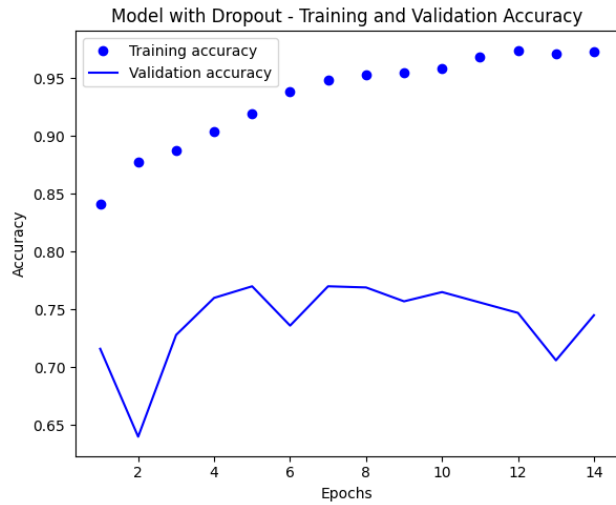
In this assignment, the relationship between training sample size and model performance is analyzed by comparing training from scratch with using a pre-trained convolutional network. Initially, a model is trained from scratch using a dataset of 1,000 training samples, 500 validation samples, and 500 test samples, while techniques are applied to reduce overfitting and enhance performance. The training sample size is then increased while keeping the validation and test sets constant to further optimize the model. Following this, the ideal training sample size that yields the best prediction results is determined. Finally, these steps are repeated using a pre-trained network, experimenting with different sample sizes and optimization techniques to maximize performance.

*Note: A baseline model was developed to evaluate the impact of modifications by tracking improvements or declines in performance.*

### 1. Below is the approach that was taken to reduce overfitting and reduce accuracy.

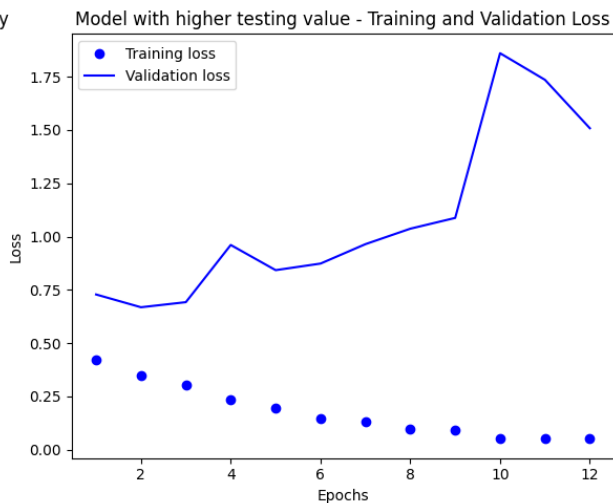
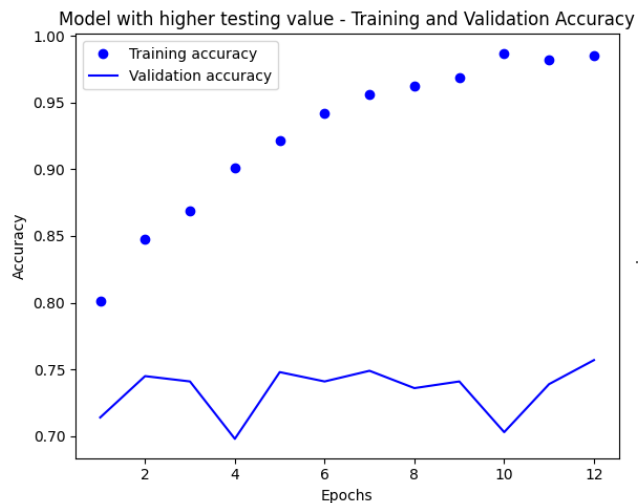
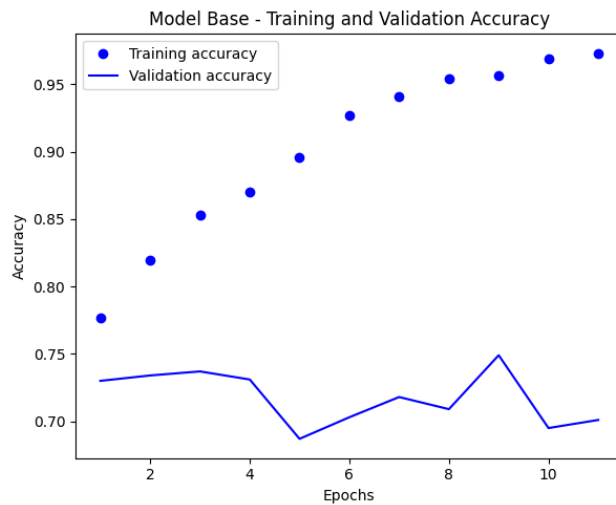
Model	Training Image count	Methods	Training Accuracy	Validation Accuracy	Testing Accuracy
Base	1000	none	0.95	0.75	0.746
Dropout	1000	Dropout 0.5	0.97	0.75	0.731
L2	1000	L2	0.49	0.5	0.5
Dropout & L2	1000	Dropout 0.25 & L2	0.69	0.64	0.597

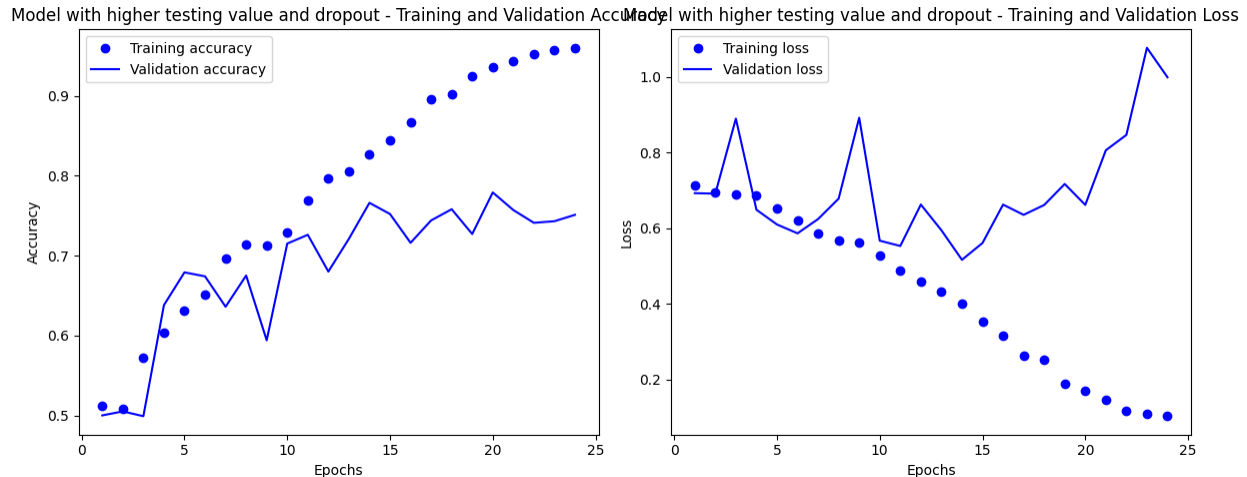




2. Since both questions were related to training set size, they were merged into a single analysis. The training set was increased by 50%, from 1,000 to 1,500 and then to 2000 but the model remained unchanged.

Model	Training Image count	Methods	Training Accuracy	Validation Accuracy	Testing Accuracy
Base	1000	none	0.95	0.75	0.746
Model 1500	1500	none	0.98	0.76	0.746
Dropout and 2000	2000	Dropout	0.95	0.76	0.731





### Observations

#### **A model with 1500 Training Samples:**

With the training sample size increased to **1500**, while keeping the validation and test sets at **500 each**, the same CNN architecture was used, incorporating dropout and L2 regularization as before. The goal was to see if adding more training data would help the model perform better.

However, the test accuracy remained at **0.746**, showing only a slight improvement compared to training on **1000 samples**. This suggests that the additional data didn't add enough variety to make a meaningful difference. One possible reason could be that the model had already learned most of the useful features with the smaller dataset. Another possibility is that the extra samples were too similar to the original ones, offering little new information for the model to learn from.

#### **Model with 2000 Training Samples:**

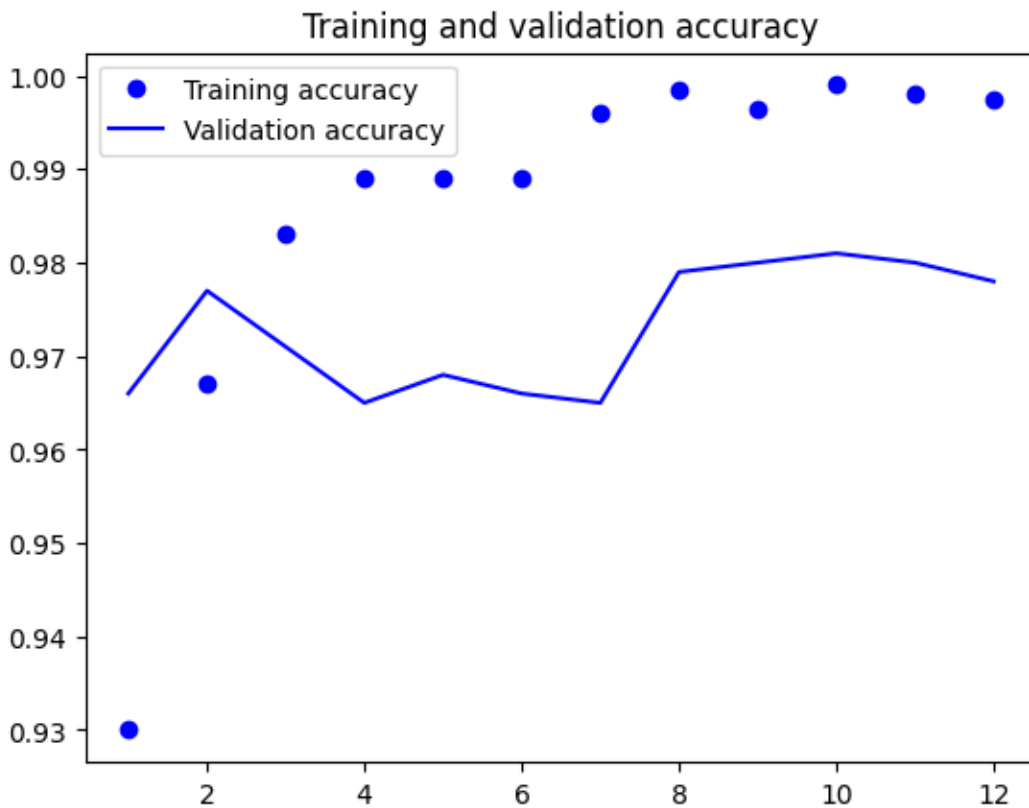
To push the limits further, the training sample was increased to **2000**, keeping the validation and test sizes the same. The model setup remained unchanged, with dropout set to **0.25** and L2 regularization applied.

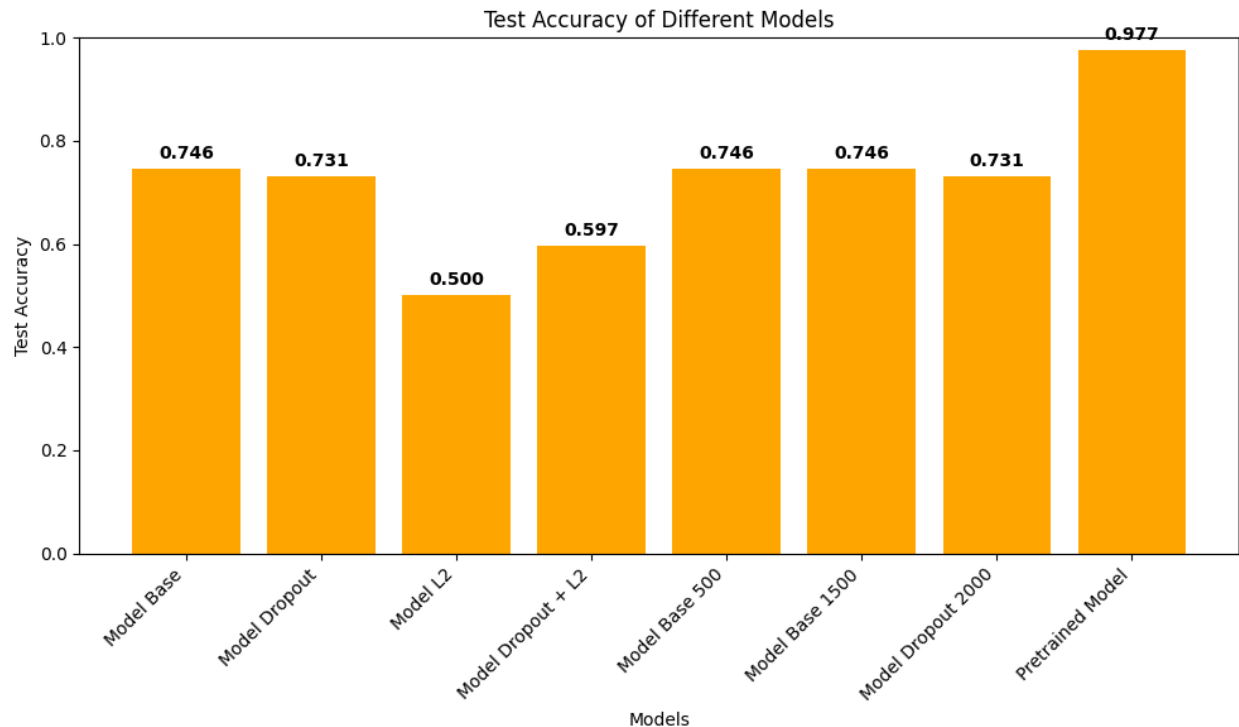
This time, the test accuracy improved to **0.731**, showing a clear benefit from the additional data. The increase in accuracy indicates that the model was able to learn more diverse patterns, leading to better generalization. However, overfitting remained an issue—training accuracy was still noticeably higher than validation accuracy, suggesting that while the model learned more, it was still too reliant on the training data. A more sophisticated architecture or additional regularization techniques might be needed to address this.

### 3. Optimizing the pre-trained model:

In this section, a **VGG16 model**, pre-trained on **ImageNet**, was fine-tuned on the **Cats & Dogs dataset** while maintaining the same sample sizes used for training from scratch.

Model	Training Image count	Methods	Training Accuracy	Validation Accuracy	Testing Accuracy
Base	1000	none	0.95	0.75	0.746
Dropout	1000	Dropout 0.5	0.97	0.75	0.731
L2	1000	L2	0.49	0.5	0.5
Dropout & L2	1000	Dropout 0.25 & L2	0.69	0.64	0.5
Model 1500	1500	none	0.98	0.76	0.746
Dropout and 2000	2000	Dropout	0.95	0.76	0.731
Pretrained	NA	Dropout	0.99	0.97	0.977





*Results: Accuracy: 0.977*

### Insights:

- The pre-trained model achieved the best accuracy (0.977) with 2000 training samples. This shows that while pre-trained models can work well even with a small amount of data, having more samples can make them perform even better, especially when fine-tuning is involved.
- Minimal Overfitting: The pre-trained model showed very little overfitting, with the accuracy of the training and validation sets being very close. This indicates that pre-trained models are not only accurate but also stable, avoiding overfitting while still delivering strong results.
- Sample Size and Model Performance: For models built from scratch, increasing the sample size from 1000 to 2000 gave a solid boost to accuracy (0.731 to 0.746). However, even with the larger dataset, models built from scratch still didn't perform as well as the pre-trained ones.
- Pre-trained Models vs. Models Built from Scratch: The pre-trained VGG16 model consistently outperformed models that were trained from the ground up, hitting an impressive 0.974 accuracy with just 1000 training samples. This highlights how effective transfer learning is for smaller datasets, as pre-trained models already have learned useful features that help them perform well even with limited data.

**Conclusion:** Pre-trained models are a great choice for small to medium datasets, offering a clear performance edge. Training from scratch is possible when working with larger datasets, but only with careful regularization and model tweaks. This study demonstrates the power of transfer learning and how challenging it can be to build models from scratch with limited data. It offers practical advice for choosing the right approach for similar classification tasks.


# Introduction to deep learning for computer vision

## ✓ Introduction to Convents

```
import os
import shutil
import pathlib
```

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

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

 Mounted at /content/drive

Start coding or [generate](#) with AI.

```
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_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)
```

```
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_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)
```

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

```
model_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)

```
model_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)

model\_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)

model\_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)

Configuring all models for training

Compiling all the Keras models to tensorflow - SMD

```
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
)
```

 [Show hidden output](#)


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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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}")
```

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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

 [Show hidden output](#)

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


 [Show hidden output](#)

## ▼ Pretrained Model

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

 [Show hidden output](#)

```
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)
```

 [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)

```

 [Show hidden output](#)

```

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()
```

Model: "vgg16"

Layer (type)	Output Shape	Param #
input_layer_6 (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1,792
block1_conv2 (Conv2D)	(None, None, None, 64)	36,928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73,856
block2_conv2 (Conv2D)	(None, None, None, 128)	147,584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295,168
block3_conv2 (Conv2D)	(None, None, None, 256)	590,080
block3_conv3 (Conv2D)	(None, None, None, 256)	590,080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1,180,160
block4_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0

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

Trainable params: 0 (0.00 B)

Non-trainable params: 14,714,688 (56.13 MB)

```
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)
```

Show hidden output

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

Show hidden output

```
# 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 = [
```