ASSIGNMENT 3

Student's name: THUY THIEN TRAM NGUYEN

ID: 811293917

SUMMARY

#Question 1: Cats_and_dogs dataset with a training sample of 1000, a validation sample of 500, and a test sample of 500

- Method:

  + First of all, creates a smaller subset of a large dataset for a cats-vs-dogs classification task by randomly selecting 500 images each for cats and dogs from the training set, and 250 each for the validation and test sets.

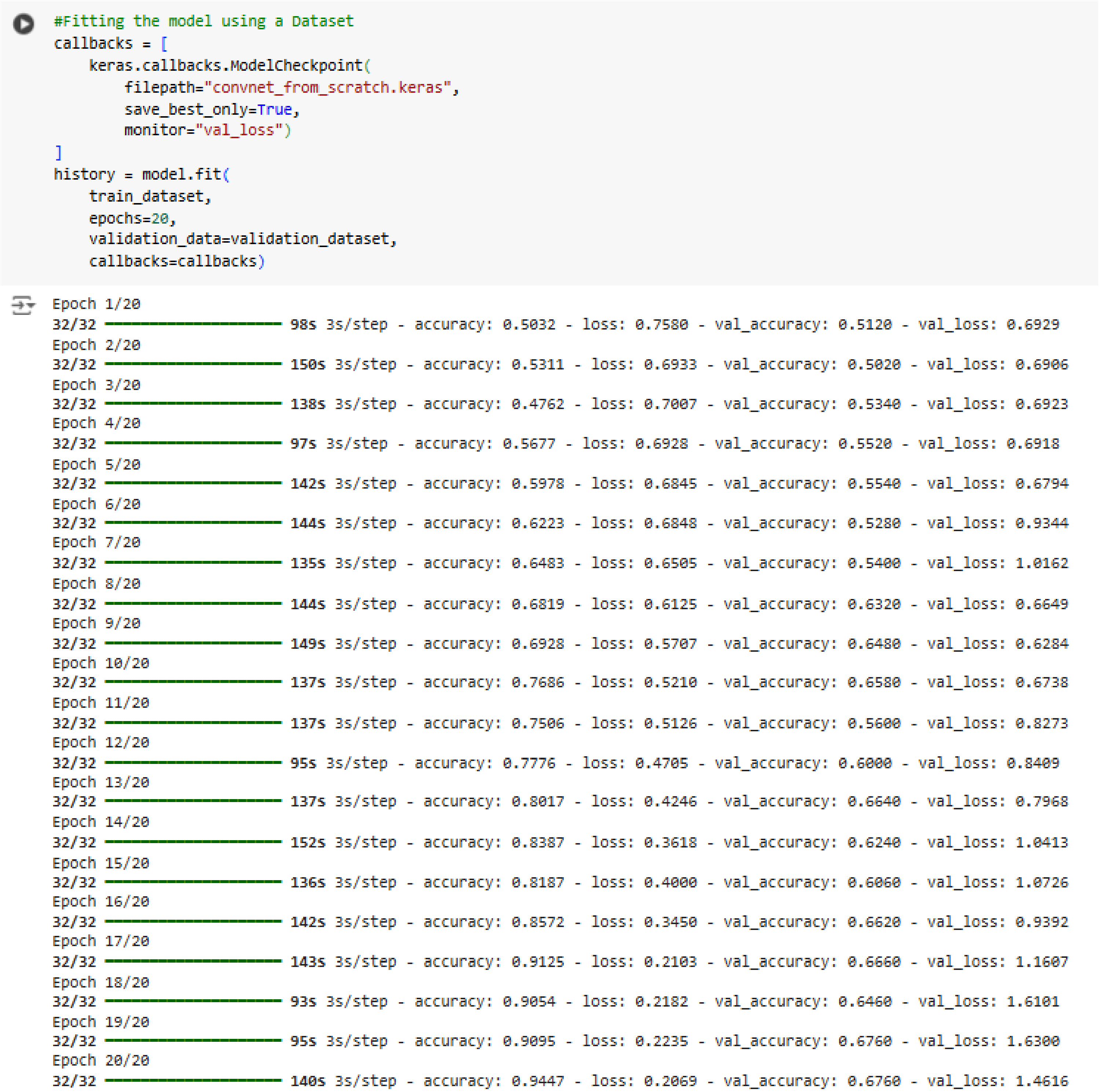
- Model's parameters:

  + Input layer: keras.Input with shape (180, 180, 3) for RGB images.
  + Normalization layer: Rescaling(1./255), which scales pixel values between 0 and 1.
  + 5 Convolutional layers (Conv2D) with **ReLU** activation, organized as follows:
  1st: 32 filters, 3x3 kernel
  2nd: 64 filters, 3x3 kernel
  3rd: 128 filters, 3x3 kernel
  4th: 256 filters, 3x3 kernel
  5th: 256 filters, 3x3 kernel
  5 Pooling layers (MaxPooling2D), each with a 2x2 pool size, following each Conv2D layer.
  + Flatten layer: Converts 2D features to a 1D vector.
  + Output layer: Dense layer with 1 unit and a **sigmoid** activation for binary classification.
  + Layers.RandomFlip("horizontal")
  + Layers.RandomRotation(0.1)
  + Layers.RandomZoom(0.2)
  + Dropout value =0.5

  ➔ In this scenario, I started with **the filters** increase **from 32 to 256** across layers to progressively capture more complex patterns. **Kernel 3x3** is effectively captures details, it allows the model to learn from a part with size 3x3 pixel of the picture to it could mainly focus on details. Each **pooling layer with a 2x2 size** progressively reduces spatial dimensions by half, simplifying the data and focusing on prominent features while controlling overfitting and minimizing computational load. **Dropout with a rate of 0.5** means that 50% of the neurons in that layer will be randomly deactivated during each training iteration

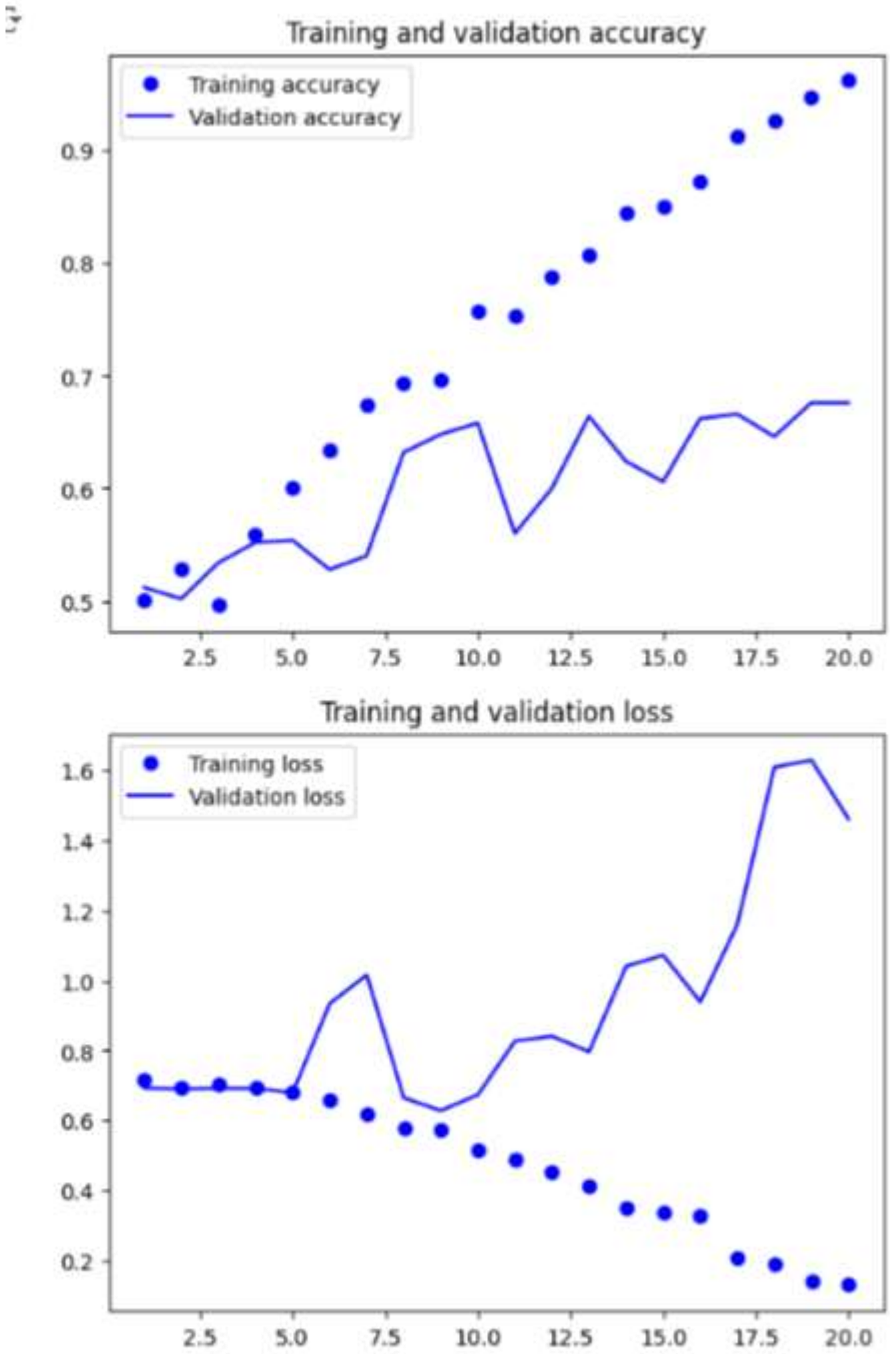
- Model's performance:

  + After fitting the model to the validation dataset, with 20 epochs, accuracy gradually increased from 50% to 94.47%, and error decreased from 0.75% to 21%. The results from the chart show that at epoch 4, validation accuracy begins to diverge and gradually improves, indicating that the model is enhancing its generalization ability on unseen data (validation set). This suggests that the model is not only memorizing the training data but also learning general features of the data. However, from the Training and Validation loss plots, it is evident that while the Training loss is sharply decreasing, the Validation loss starts to increase at epoch 5, indicating potential overfitting. Currently, the learning rate is set to 1e-5, so we may consider reducing the learning rate further to allow the model to learn slowly and carefully, focusing on relevant features.



```
#Fitting the model using a Dataset
callbacks = [
    keras.callbacks.ModelCheckpoint(
        filepath="convnet_from_scratch.keras",
        save_best_only=True,
        monitor="val_loss")
]
history = model.fit(
    train_dataset,
    epochs=20,
    validation_data=validation_dataset,
    callbacks=callbacks)
```

```
Epoch 1/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 98s 3s/step - accuracy: 0.5032 - loss: 0.7580 - val_accuracy: 0.5120 - val_loss: 0.6929
Epoch 2/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 150s 3s/step - accuracy: 0.5311 - loss: 0.6933 - val_accuracy: 0.5020 - val_loss: 0.6906
Epoch 3/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 138s 3s/step - accuracy: 0.4762 - loss: 0.7007 - val_accuracy: 0.5340 - val_loss: 0.6923
Epoch 4/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 97s 3s/step - accuracy: 0.5677 - loss: 0.6928 - val_accuracy: 0.5520 - val_loss: 0.6918
Epoch 5/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 142s 3s/step - accuracy: 0.5978 - loss: 0.6845 - val_accuracy: 0.5540 - val_loss: 0.6794
Epoch 6/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 144s 3s/step - accuracy: 0.6223 - loss: 0.6848 - val_accuracy: 0.5280 - val_loss: 0.9344
Epoch 7/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 135s 3s/step - accuracy: 0.6483 - loss: 0.6505 - val_accuracy: 0.5400 - val_loss: 1.0162
Epoch 8/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 144s 3s/step - accuracy: 0.6819 - loss: 0.6125 - val_accuracy: 0.6320 - val_loss: 0.6649
Epoch 9/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 149s 3s/step - accuracy: 0.6928 - loss: 0.5707 - val_accuracy: 0.6480 - val_loss: 0.6284
Epoch 10/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 137s 3s/step - accuracy: 0.7686 - loss: 0.5210 - val_accuracy: 0.6580 - val_loss: 0.6738
Epoch 11/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 137s 3s/step - accuracy: 0.7506 - loss: 0.5126 - val_accuracy: 0.5600 - val_loss: 0.8273
Epoch 12/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 95s 3s/step - accuracy: 0.7776 - loss: 0.4705 - val_accuracy: 0.6000 - val_loss: 0.8409
Epoch 13/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 137s 3s/step - accuracy: 0.8017 - loss: 0.4246 - val_accuracy: 0.6640 - val_loss: 0.7968
Epoch 14/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 152s 3s/step - accuracy: 0.8387 - loss: 0.3618 - val_accuracy: 0.6240 - val_loss: 1.0413
Epoch 15/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 136s 3s/step - accuracy: 0.8187 - loss: 0.4000 - val_accuracy: 0.6060 - val_loss: 1.0726
Epoch 16/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 142s 3s/step - accuracy: 0.8572 - loss: 0.3450 - val_accuracy: 0.6620 - val_loss: 0.9392
Epoch 17/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 143s 3s/step - accuracy: 0.9125 - loss: 0.2103 - val_accuracy: 0.6660 - val_loss: 1.1607
Epoch 18/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 93s 3s/step - accuracy: 0.9054 - loss: 0.2182 - val_accuracy: 0.6460 - val_loss: 1.6101
Epoch 19/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 95s 3s/step - accuracy: 0.9095 - loss: 0.2235 - val_accuracy: 0.6760 - val_loss: 1.6300
Epoch 20/20
32/32 ━━━━━━━━━━━━━━━━━━━━ 140s 3s/step - accuracy: 0.9447 - loss: 0.2069 - val_accuracy: 0.6760 - val_loss: 1.4616
```



#Question 2: Cats_and_dogs dataset with a training sample of 1500, a validation sample of 500, and a test sample of 500

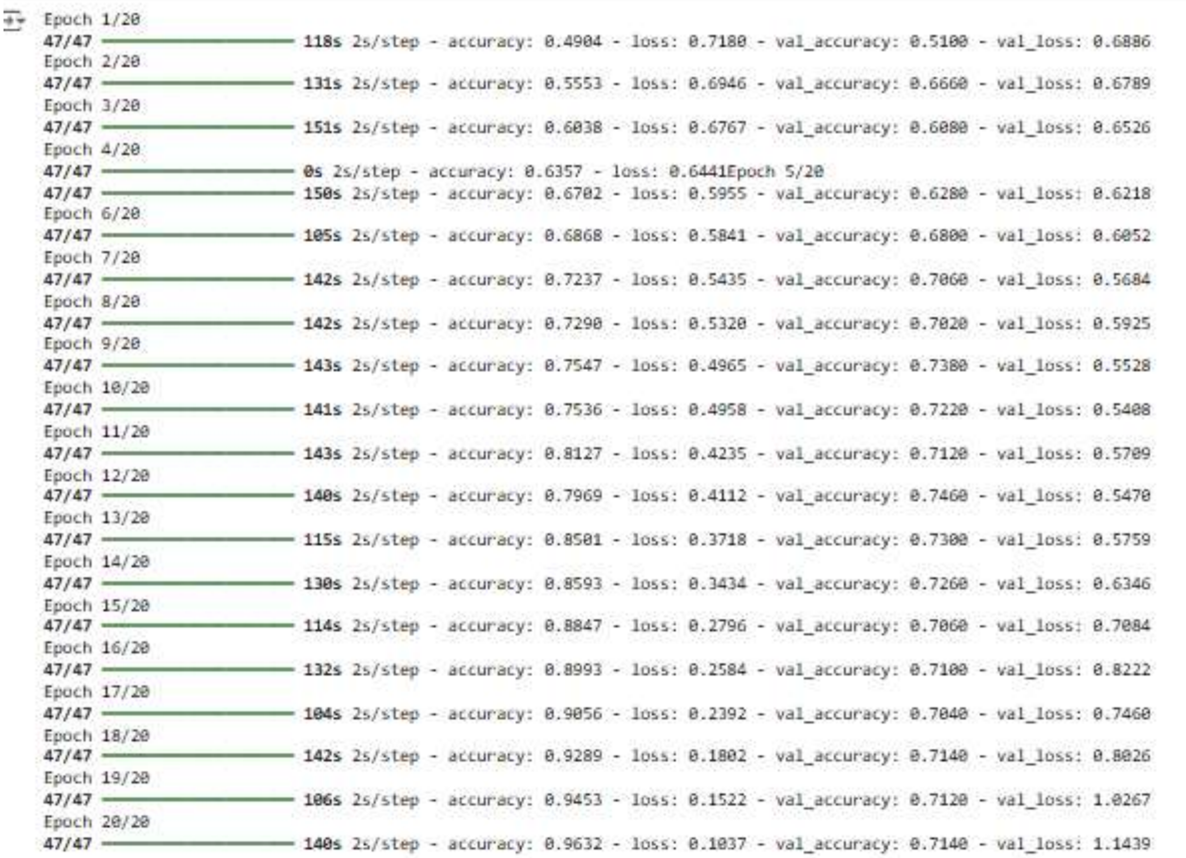
- Method:
  + This time I splited the dataset into 1500 training, 500 variables for each validation and test set
- Model:
  + Other parameters are kept the same. However, I changed the kernel into 2x2 size

+ Layers.RandomRotation is 0.2

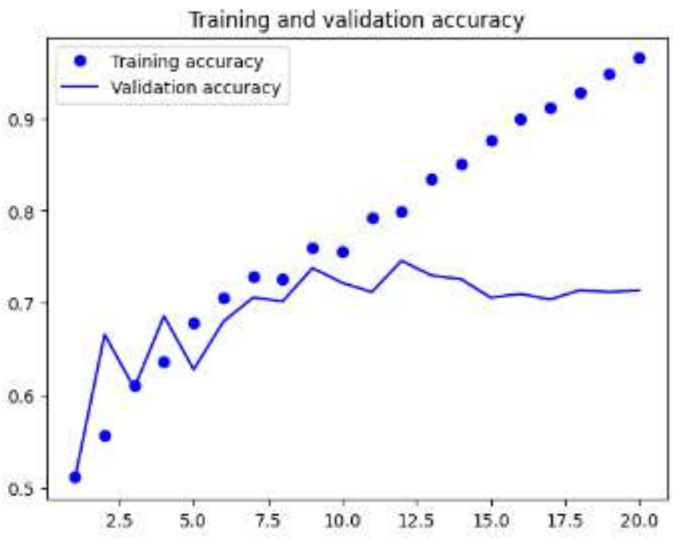
+ Dropout value is 0.5

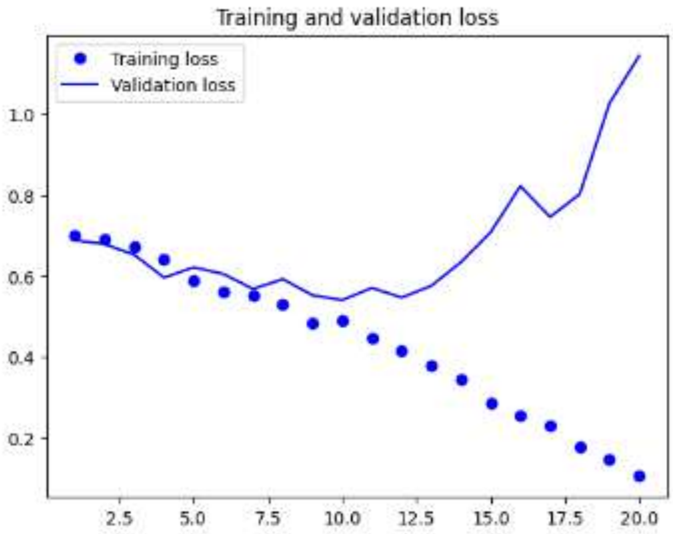
- Model’s performance:

  + The result depicts that both training and validation accuracy gradually improved, and validation loss is slightly getting better as the value is smaller than Scenario 1.



```
Epoch 1/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 118s 2s/step - accuracy: 0.4904 - loss: 0.7180 - val_accuracy: 0.5100 - val_loss: 0.6886
Epoch 2/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 131s 2s/step - accuracy: 0.5553 - loss: 0.6946 - val_accuracy: 0.6660 - val_loss: 0.6789
Epoch 3/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 151s 2s/step - accuracy: 0.6038 - loss: 0.6767 - val_accuracy: 0.6080 - val_loss: 0.6526
Epoch 4/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 0s 2s/step - accuracy: 0.6357 - loss: 0.6441Epoch 5/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 150s 2s/step - accuracy: 0.6702 - loss: 0.5955 - val_accuracy: 0.6280 - val_loss: 0.6218
Epoch 6/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 105s 2s/step - accuracy: 0.6868 - loss: 0.5841 - val_accuracy: 0.6800 - val_loss: 0.6052
Epoch 7/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.7237 - loss: 0.5435 - val_accuracy: 0.7060 - val_loss: 0.5684
Epoch 8/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.7290 - loss: 0.5320 - val_accuracy: 0.7020 - val_loss: 0.5925
Epoch 9/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 143s 2s/step - accuracy: 0.7547 - loss: 0.4965 - val_accuracy: 0.7380 - val_loss: 0.5528
Epoch 10/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 141s 2s/step - accuracy: 0.7536 - loss: 0.4958 - val_accuracy: 0.7220 - val_loss: 0.5408
Epoch 11/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 143s 2s/step - accuracy: 0.8127 - loss: 0.4235 - val_accuracy: 0.7120 - val_loss: 0.5709
Epoch 12/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 140s 2s/step - accuracy: 0.7969 - loss: 0.4112 - val_accuracy: 0.7460 - val_loss: 0.5470
Epoch 13/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 115s 2s/step - accuracy: 0.8501 - loss: 0.3718 - val_accuracy: 0.7300 - val_loss: 0.5759
Epoch 14/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 130s 2s/step - accuracy: 0.8593 - loss: 0.3434 - val_accuracy: 0.7260 - val_loss: 0.6346
Epoch 15/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 114s 2s/step - accuracy: 0.8847 - loss: 0.2796 - val_accuracy: 0.7060 - val_loss: 0.7084
Epoch 16/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 132s 2s/step - accuracy: 0.8993 - loss: 0.2584 - val_accuracy: 0.7100 - val_loss: 0.8222
Epoch 17/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 104s 2s/step - accuracy: 0.9056 - loss: 0.2392 - val_accuracy: 0.7040 - val_loss: 0.7460
Epoch 18/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.9289 - loss: 0.1802 - val_accuracy: 0.7140 - val_loss: 0.8026
Epoch 19/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 106s 2s/step - accuracy: 0.9453 - loss: 0.1522 - val_accuracy: 0.7120 - val_loss: 1.0267
Epoch 20/20
47/47 ━━━━━━━━━━━━━━━━━━━━ 140s 2s/step - accuracy: 0.9632 - loss: 0.1037 - val_accuracy: 0.7140 - val_loss: 1.1439
```





+ And then, after increased the size of training dataset, it can seen that the result of training accuracy improved from 0.64 (in question 1) to 0.740. Moreover, kernel 2x2 would create less parameters than comparing with kernel 3x3 earlier. The main reason because the more parameters, the more complexity that algorithm has to store and learn. Here is the visualization after an adjustment.

#Question 3: Cats_and_dogs dataset with a training sample of 1200 (which is 700 dogs and 500 cats), a validation sample of 500, and a test sample of 500

- Method:

+ This time I want to test the accuracy of training set based on the different in dogs sample and cats sample size, which is divided into 700 dogs and 500 cats. However, the validation and test set are the same.

- Model:

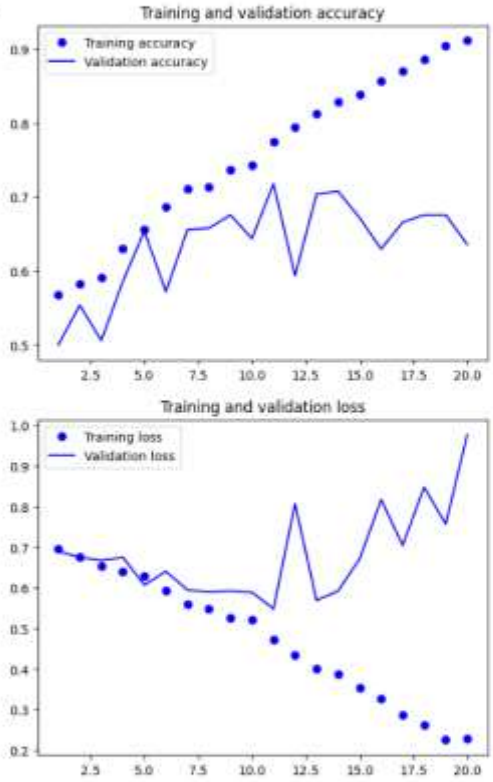
+ Input layer: keras.Input with shape (180, 180, 3) for RGB images.
+ Normalization layer: Rescaling(1./255), which scales pixel values between 0 and 1.
+ 5 Convolutional layers (Conv2D) with **ReLU** activation, organized as follows:
1st: 32 filters, 2x2 kernel

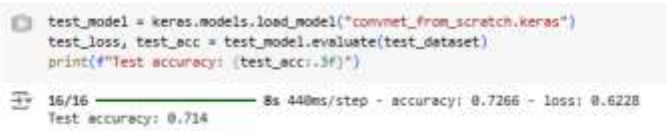
2nd: 64 filters, 2x2 kernel
3rd: 128 filters, 2x2 kernel
4th: 256 filters, 2x2 kernel
5th: 256 filters, 2x2 kernel
5 Pooling layers (MaxPooling2D), each with a 2x2 pool size, following each Conv2D layer.
+ Flatten layer: Converts 2D features to a 1D vector.
+ Output layer: Dense layer with 1 unit and a **sigmoid** activation for binary classification.
+ Layers.RandomFlip("horizontal")
+ Layers.RandomRotation(0.1)
+ Layers.RandomZoom(0.2)
+ Dropout value =0.8

➔ In this scenario, I decreased kernel size into **Kernel 2x2** which helps model to learn overly complex and localized features. Each **pooling layer with a 2x2 size** progressively reduces spatial dimensions by half, simplifying the data and focusing on prominent features while controlling overfitting and minimizing computational load. **Dropout with a rate of 0.8** means that 80% of the neurons in that layer will be randomly deactivated during each training iteration.

- Model's performance:

+ Compared to the previous questions, the accuracy of the training and test sets has somewhat declined. The possible reason could be that reducing the kernel size from 3×3 to 2×2 limits the model's ability to detect large or complex spatial patterns in the data. This reduction may decrease the accuracy of the training set, as the model may struggle to capture the full range of details and dependencies within the data. Additionally, increasing the dropout rate to 0.8 means that more neurons are being deactivated in each training iteration, which lowers the accuracy of the training set because the model is training on a smaller subset of neurons.





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

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 8s 440ms/step - accuracy: 0.7266 - loss: 0.6228
Test accuracy: 0.714
```

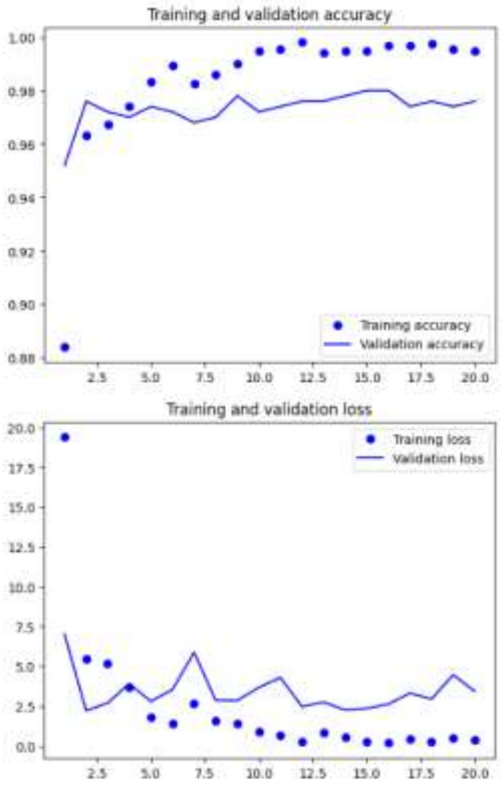
#Question 4: Cats_and_dogs dataset with a training sample of 1500, a validation sample of 500, and a test sample of 500, but using pretrained network and compare the accuracy with the model is trained from scratch.

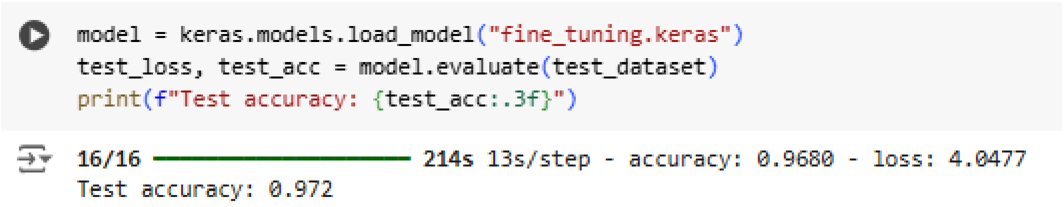
- Method:

  + The number of variables in the training, validation, and test sets is 1500, 500, and 500, respectively. The training method used is a Pretrained network, specifically VGG16.

- Model's performance:

  + The results show a remarkable improvement when using the VGG16 model, which is pretrained on the ImageNet dataset, encompassing over a million images with various angles. This pretraining allows our cat-dog dataset, even with augmentation applied, to achieve high accuracy. While data augmentation alone was insufficient to reach over 90% accuracy, using the pretrained network brought the results to a 97.2% accuracy on the test set. Additionally, the loss function gap between the validation and training datasets has narrowed.





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

```
16/16 ━━━━━━━━━━━━━━━━━━━━ 214s 13s/step - accuracy: 0.9680 - loss: 4.0477
Test accuracy: 0.972
```