Advanced Data Mining Project - Cats-vs-Dogs image classification

Szymon Lepianka

Description of the solved problem

The dataset used for this task is the "Cats-vs-Dogs Images" dataset available on Kaggle (https://www.kaggle.com/datasets/shaunthesheep/microsoft-catsvsdogs-dataset). The dataset contains 25,000 images of cats and dogs. The images are all in JPG format and have varying sizes.

The problem that was solved is a binary image classification problem of classifying images of dogs and cats. The goal was to train a model that can accurately distinguish between images of dogs and images of cats. It is a great dataset to practice building a simple image classifier using deep learning techniques.

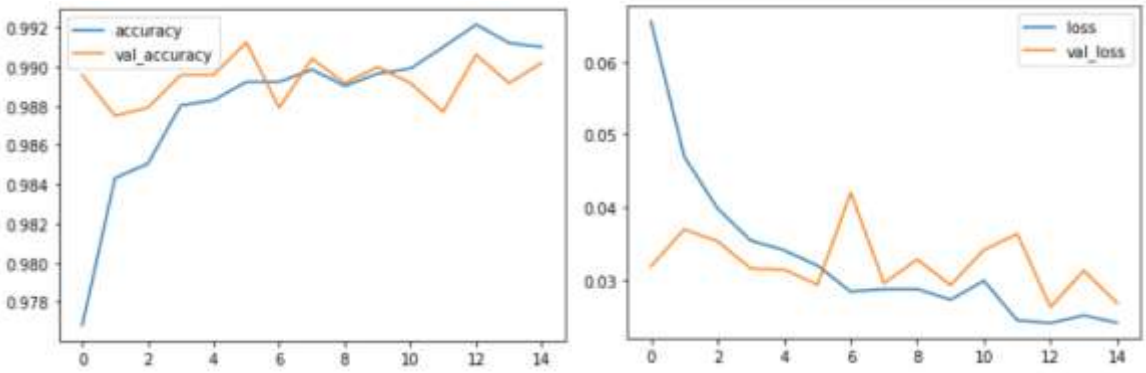
To solve this problem, a convolutional neural network (CNN) model was used. CNNs are a type of neural network that are particularly well suited for image classification tasks. They consist of multiple layers, including convolutional layers that extract features from the images, and fully connected layers that make the final predictions.

In this case, the dataset was used to train the CNN model to learn to identify the features that distinguish cats from dogs in images. The model was then tested on the test set, to evaluate its performance.

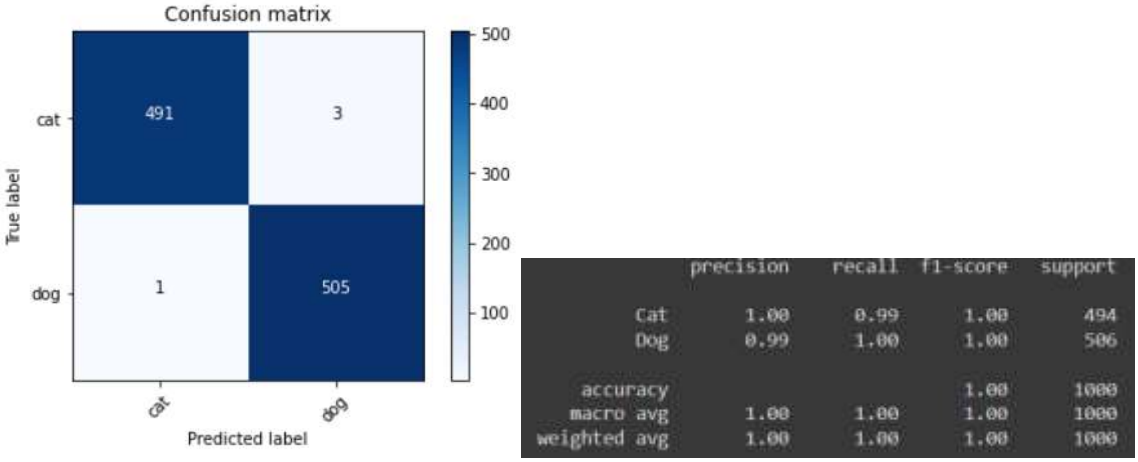
The best result was made using an EfficientNetB4 model (in source code (.ipynb) in section “Model 6”) which is a state-of-the-art architecture. The EfficientNetB4 model was pretrained on a large dataset, allowing it to learn general features of images before being fine-tuned on the “Cats vs. Dogs” dataset.

Additionally, data augmentation was used to increase the amount of training data available. Data augmentation techniques such as rotation, flipping, and zoom were applied to the images to artificially increase the number of training examples. This helped to improve the performance of the model.

The accuracy of the model on the test set was 0.9959999918937683. Performance of the model was evaluated by visualizing its learning curve, confusion matrix, classification report and misclassified objects.



Plot of the accuracy and loss of the model over time.



	precision	recall	f1-score	support
Cat	1.00	0.99	1.00	494
Dog	0.99	1.00	1.00	506
accuracy			1.00	1000
macro avg	1.00	1.00	1.00	1000
weighted avg	1.00	1.00	1.00	1000

Confusion matrix and classification report of the best model.

Reference to the source of the data.

The “Cats-vs-Dogs” dataset is a popular dataset used for image classification tasks and is widely used in machine learning research and development. The dataset is sourced from Kaggle, which is a platform for machine learning competitions and a place to find and share datasets. The dataset is publicly available for use by the machine learning community. The dataset contains 25,000 images of cats and dogs. The images are of various sizes and have been labeled as either "cat" or "dog." The goal of the dataset is to accurately predict the class of an image based on its features.

Among the cat pictures, the picture named "666.jpg" was empty. It got deleted during preprocessing. The same was true for a photo of a dog named "11702.jpg".

Each photo has its ID in the filename. During preprocessing, the prefix "cat." or "dog." was added to the filenames, respectively.

Random images of cats and dogs.

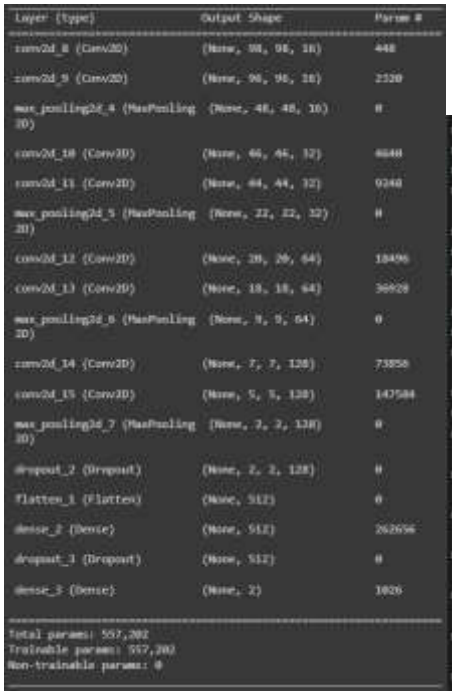
Description of solution of the problem.

The dataset of images was splitted into three sets: training, validation, and test. The dataset was first stored in a dataframe called *image_df*, which contains file names and their categories. The dataset was splitted by using the *train_test_split* function, which randomly splits the dataset into the desired number of sets.

ImageDataGenerator and *flow_from_dataframe* were used to load the images. The data generators were created for the training, validation, and test sets of images. It was used to load images and their labels into memory in batches during the training process. All images were resized to 100x100 pixels. The *batch_size* variable is set to 64, which means that 64 images will be loaded into memory at a time.

In the end, 6 different models were created.

Model 1 (in source code “Model 0”)



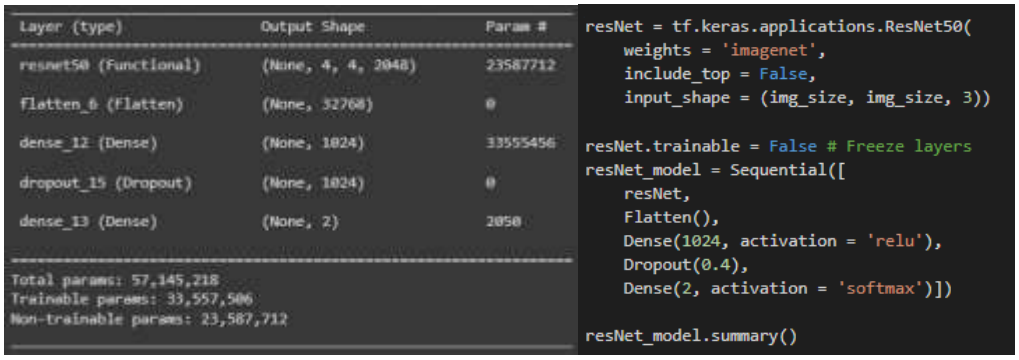
Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 98, 98, 16)	448
conv2d_9 (Conv2D)	(None, 96, 96, 16)	2320
max_pooling2d_4 (MaxPooling 2D)	(None, 48, 48, 16)	0
conv2d_10 (Conv2D)	(None, 46, 46, 32)	4640
conv2d_11 (Conv2D)	(None, 44, 44, 32)	9248
max_pooling2d_5 (MaxPooling 2D)	(None, 22, 22, 32)	0
conv2d_12 (Conv2D)	(None, 20, 20, 64)	18496
conv2d_13 (Conv2D)	(None, 18, 18, 64)	36928
max_pooling2d_6 (MaxPooling 2D)	(None, 9, 9, 64)	0
conv2d_14 (Conv2D)	(None, 7, 7, 128)	73856
conv2d_15 (Conv2D)	(None, 5, 5, 128)	147584
max_pooling2d_7 (MaxPooling 2D)	(None, 2, 2, 128)	0
dropout_2 (Dropout)	(None, 2, 2, 128)	0
flatten_1 (Flatten)	(None, 512)	0
dense_2 (Dense)	(None, 512)	262656
dropout_3 (Dropout)	(None, 512)	0
dense_3 (Dense)	(None, 2)	1026

Total params: 557,282
Trainable params: 557,282
Non-trainable params: 0

```
model0 = Sequential()
model0.add(Conv2D(16, (3,3), activation = 'relu', input_shape = (img_size, img_size, 3)))
model0.add(Conv2D(16, (3,3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))

model0.add(Conv2D(32, (3,3), activation = 'relu'))
model0.add(Conv2D(32, (3,3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))

model0.add(Conv2D(64, (3,3), activation = 'relu'))
model0.add(Conv2D(64, (3,3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))

model0.add(Conv2D(128, (3,3), activation = 'relu'))
model0.add(Conv2D(128, (3,3), activation = 'relu'))
model0.add(MaxPooling2D(pool_size = (2,2), strides=(2,2)))

model0.add(Dropout(0.3))

model0.add(Flatten())

model0.add(Dense(512, activation = 'relu'))

model0.add(Dropout(0.5))

model0.add(Dense(2, activation = 'softmax'))

model0.summary()
```

A sequential model in Keras with several layers was defined. The model is composed of a series of Convolutional Neural Network (CNN) layers, pooling layers, dropout layers, and fully connected layers. It is designed to take image inputs and output a probability of the image belonging to each class. It has several dropout layers to prevent overfitting and a final softmax activation function for the output layer.

Model 2



Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 4, 4, 2048)	23587712
flatten_6 (Flatten)	(None, 32768)	0
dense_12 (Dense)	(None, 1024)	33555456
dropout_15 (Dropout)	(None, 1024)	0
dense_13 (Dense)	(None, 2)	2050

Total params: 57,145,218
Trainable params: 33,557,506
Non-trainable params: 23,587,712

```
resNet = tf.keras.applications.ResNet50(
    weights = 'imagenet',
    include_top = False,
    input_shape = (img_size, img_size, 3))

resNet.trainable = False # Freeze layers
resNet_model = Sequential([
    resNet,
    Flatten(),
    Dense(1024, activation = 'relu'),
    Dropout(0.4),
    Dense(2, activation = 'softmax')])

resNet_model.summary()
```

This model is a deep learning model that uses the ResNet50 architecture as its base. The ResNet50 model is pre-trained on the ImageNet dataset and its layers are frozen during training. The model has several fully connected layers and dropout layers added on top of the ResNet50 model. The final output layer has 2 units and a softmax activation function which outputs the probability of the image belonging to each class.

Model 3

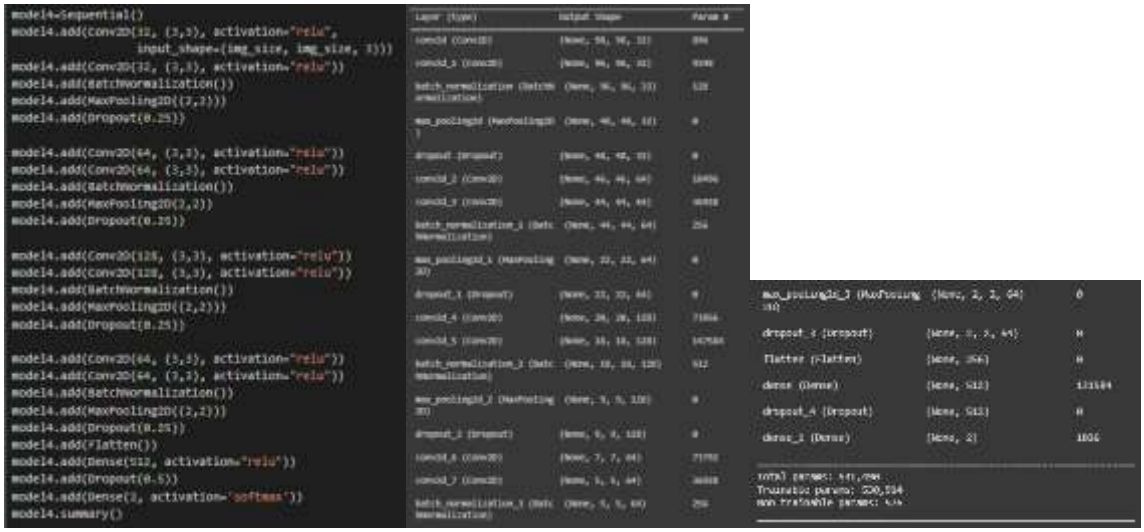
```
model3 = Sequential()

model3.add(Conv2D(32, (3, 3), activation='relu',
                  input_shape=(img_size, img_size, 3)))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(64, (3, 3), activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Conv2D(128, (3, 3), activation='relu'))
model3.add(BatchNormalization())
model3.add(MaxPooling2D(pool_size=(2, 2)))
model3.add(Dropout(0.25))

model3.add(Flatten())
model3.add(Dense(512, activation='relu'))
model3.add(BatchNormalization())
model3.add(Dropout(0.5))
model3.add(Dense(2, activation='softmax'))

model3.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_46 (Conv2D)           (None, 98, 98, 32)        896

batch_normalization_9 (Batc  (None, 98, 98, 32)        128
hNormalization)

max_pooling2d_27 (MaxPoolin  (None, 49, 49, 32)        0
g2D)

dropout_16 (Dropout)         (None, 49, 49, 32)        0

conv2d_47 (Conv2D)           (None, 47, 47, 64)        18496

batch_normalization_10 (Bat  (None, 47, 47, 64)        256
chNormalization)

max_pooling2d_28 (MaxPoolin  (None, 23, 23, 64)        0
g2D)

dropout_17 (Dropout)         (None, 23, 23, 64)        0

conv2d_48 (Conv2D)           (None, 21, 21, 128)       73856

batch_normalization_11 (Bat  (None, 21, 21, 128)       512
chNormalization)

max_pooling2d_29 (MaxPoolin  (None, 10, 10, 128)       0
g2D)

dropout_18 (Dropout)         (None, 10, 10, 128)       0

flatten_7 (Flatten)          (None, 12800)             0

dense_14 (Dense)             (None, 512)               6554112

batch_normalization_12 (Bat  (None, 512)               2048
chNormalization)

dropout_19 (Dropout)         (None, 512)               0

dense_15 (Dense)             (None, 2)                 1026

=================================================================
Total params: 6,651,330
Trainable params: 6,649,858
Non-trainable params: 1,472
```

Model 3 is convolutional neural network (CNN) and is composed of several convolutional layers, each followed by a batch normalization layer, a max pooling layer, and a dropout layer. The max pooling layer is used to down-sample the feature maps, while the dropout layer is used to prevent overfitting.

Compared to “Model 0”, “Model 3” has a different number of filters in the convolutional layers and it uses batch normalization, which is used to standardize the inputs to the network, which can help to improve the stability and speed up the training of the model.

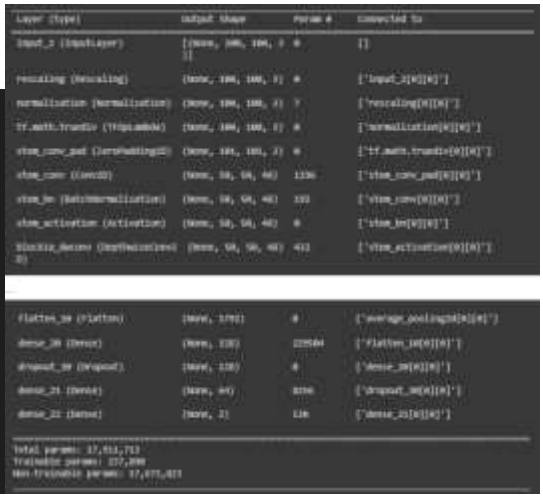
Model 4



This model is similar to “Model 3” in that it uses a combination of convolutional layers, batch normalization, max pooling and dropout layers. However, it has an additional block of two convolutional layers and one max pooling layer before the final flatten and dense layers. This may result in more complex features being learned by the model, potentially leading to better performance.

Model 5

```
base_en_model = EfficientNetB4(weights="imagenet",
                               include_top=False,
                               input_shape=(img_size, img_size, 3),
                               classes=2)

for layer in base_en_model.layers:
        layer.trainable = False

output_model = base_en_model.output
output_model = AveragePooling2D(pool_size=(3,3))(output_model)
output_model = Flatten()(output_model)
output_model = Dense(128, activation="relu")(output_model)
output_model = Dropout(0.4)(output_model)
output_model = Dense(64, activation="relu")(output_model)
output_model = Dense(2, activation="softmax")(output_model)
model5 = Model(inputs=base_en_model.input, outputs=output_model)
model5.summary()
```



This is a Keras model that utilizes the EfficientNetB4 architecture with pre-trained weights from the ImageNet dataset. The model is also set to not train the layers from the EfficientNetB4 architecture. After the base model, it applies an AveragePooling2D layer with a pool size of (3,3), then it flattens the output and applies two dense layers with ReLU activation function, a dropout layer with a dropout rate of 0.4 and a final dense layer with a softmax activation function for 2 classes.

Model 6

The model remained unchanged (from “Model 5”), but data augmentation was added. Image size (for *ImageDataGenerator*) was set to 224. Data generators for training and validation apply a set of data augmentation techniques to the images. The techniques include rotating the images by a random degree between 0 and 30, zooming in by a random factor between 0.85 and 1.15, shifting the width and height of the images by a random factor between -0.2 and 0.2, applying a random shear transformation between -0.15 and 0.15, and flipping the images horizontally with a 50% probability. The third generator doesn't apply any data augmentation to the images in the test dataset.

Compiling and fitting models

Models are compiled using the **categorical_crossentropy** loss function and the **Adam** optimizer, and it's measuring accuracy during the training process. The loss function is a way to measure the performance of a machine learning model, specifically how well it is able to predict the expected output.
categorical_crossentropy is typically used for classification problems, where the model must assign a probability to each of several possible classes. It measures the difference

between the predicted class probabilities and the true class. The smaller the loss value, the better the model is at predicting the correct class.

Adam is an optimization algorithm used to update the model's weights. Adam uses the gradient of the loss function to update the model's weights and it also takes into account the past gradients to adjust the learning rate. It is a popular optimization algorithm because it's computationally efficient and it generally works well with a wide range of different types of problems.

An **EarlyStopping** callback is also set up, which monitors the validation loss and stops the training if the validation loss does not improve for a number of epochs.

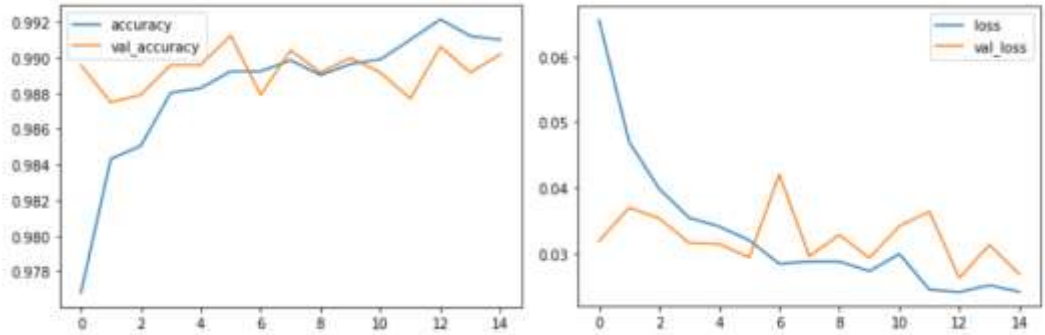
The model is then trained for a number of epochs using the fit method, with the training data and validation data provided. The batch size is used to divide the data into smaller chunks for training and validation steps, by taking the floor division of the number of images by the batch size.

Results

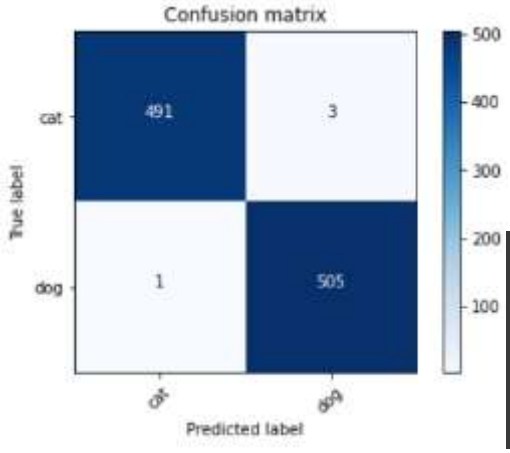
(From the best to the worst.)

Best result (Model 6)

The best results were achieved using **model 6**.
Accuracy = 0.9959999918937683.
Thanks to increasing image size and adding data augmentation, the model performed better than “Model 5” (smaller images and no data augmentation, same model architecture).

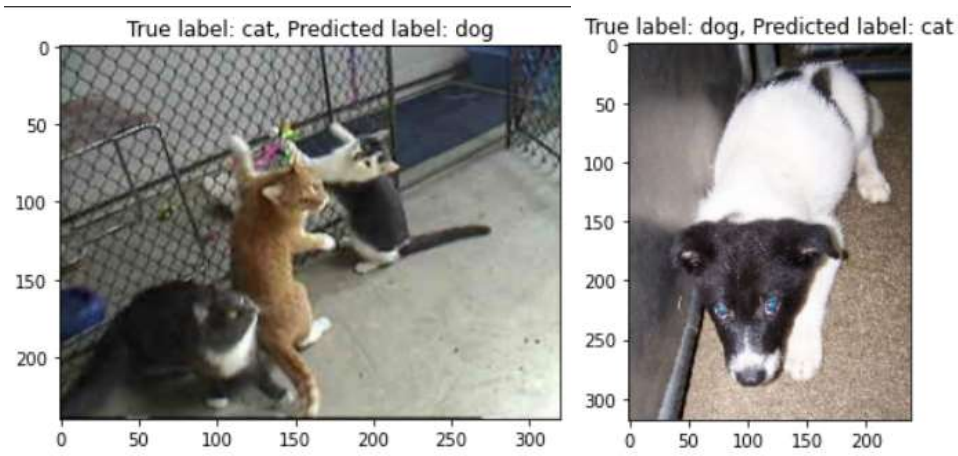


Plot of the accuracy and loss of the model over time.



	precision	recall	f1-score	support
Cat	1.00	0.99	1.00	494
Dog	0.99	1.00	1.00	506
accuracy			1.00	1000
macro avg	1.00	1.00	1.00	1000
weighted avg	1.00	1.00	1.00	1000

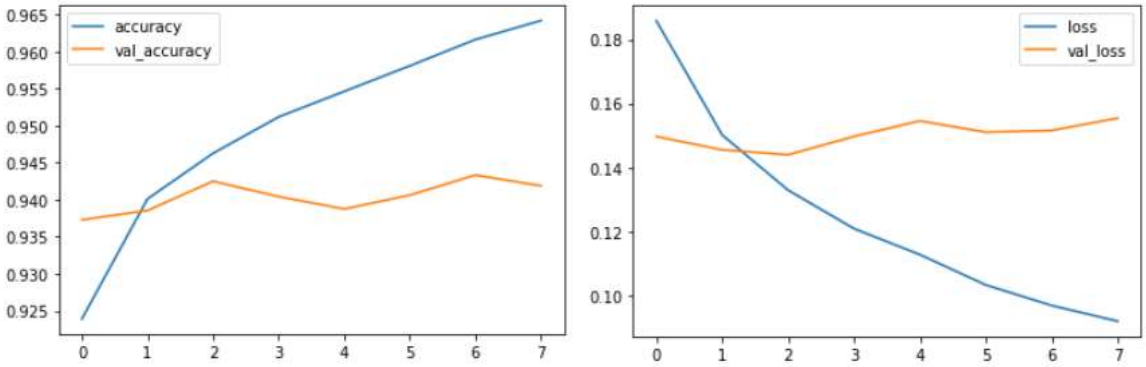
Confusion matrix and classification report of the best model.



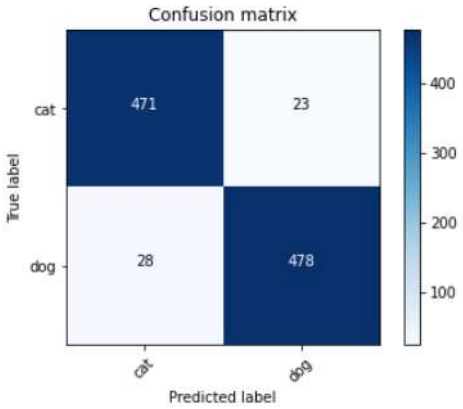
Two misclassified objects

2nd best result (Model 5)

Second best results were achieved using **model 5**.
Accuracy = 0.9490000009536743.
Using EfficientNetB4 performed better than ResNet50 (“Model 2”).



Plot of the accuracy and loss of the model over time.



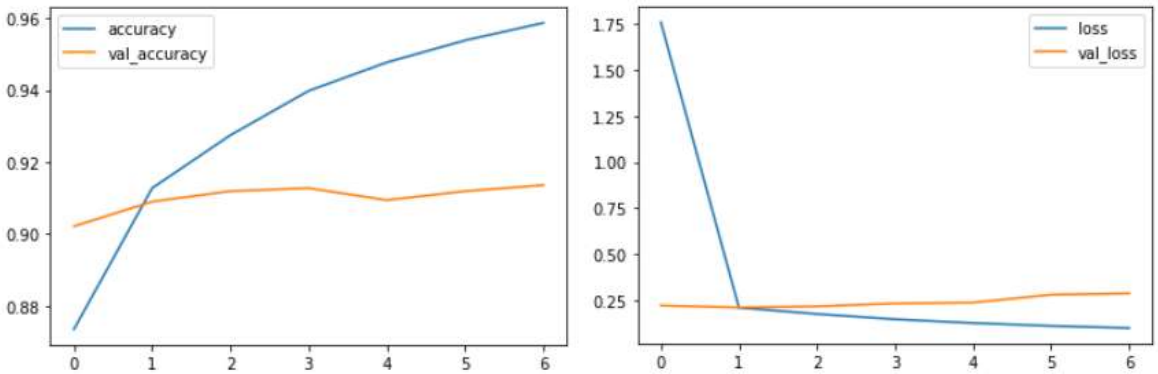
	precision	recall	f1-score	support
Cat	0.94	0.95	0.95	494
Dog	0.95	0.94	0.95	506
accuracy			0.95	1000
macro avg	0.95	0.95	0.95	1000
weighted avg	0.95	0.95	0.95	1000

Confusion matrix and classification report of the second best model.

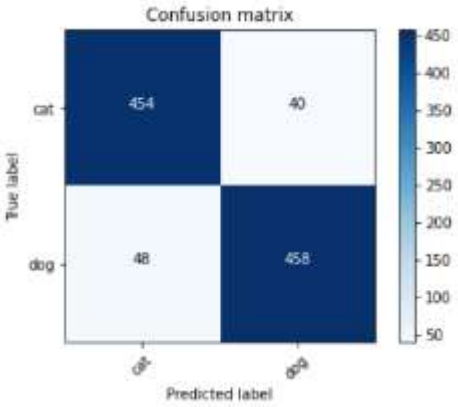
Two misclassified objects

3rd best result (Model 2)

Third best results were achieved using **model 2**.
Accuracy = 0.9120000004768372.
By using transfer learning (ResNet50), the achieved results were better than without it (better than “Model 4”, “Model 1” and “Model 3”).

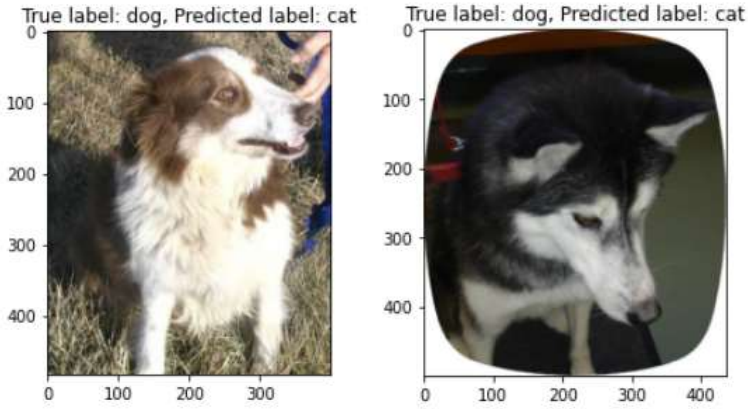


Plot of the accuracy and loss of the model over time.



	precision	recall	f1-score	support
Cat	0.90	0.92	0.91	494
Dog	0.92	0.91	0.91	506
accuracy			0.91	1000
macro avg	0.91	0.91	0.91	1000
weighted avg	0.91	0.91	0.91	1000

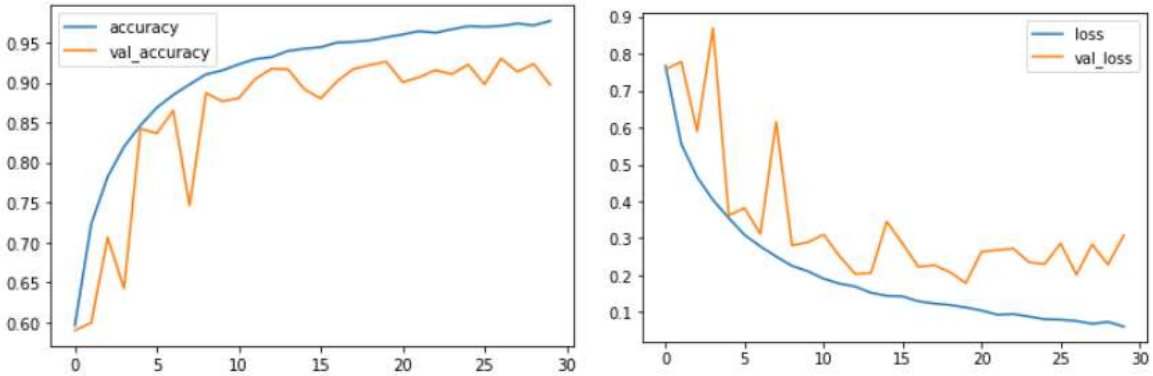
Confusion matrix and classification report of the third best model.



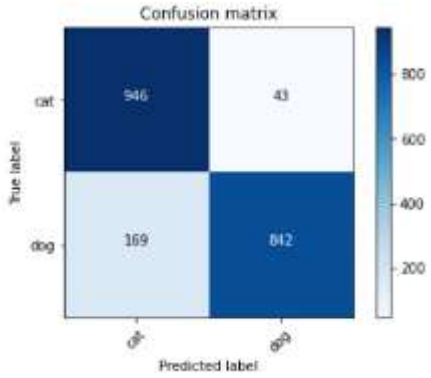
Two misclassified objects

4th best result (Model 4)

Forth best results were achieved using **model 4**.
Accuracy = 0.8939999938011169.
This model is the result of combining “Model 1” and “Model 3”, which results weren’t satisfactory. This model performed better than mentioned ones.

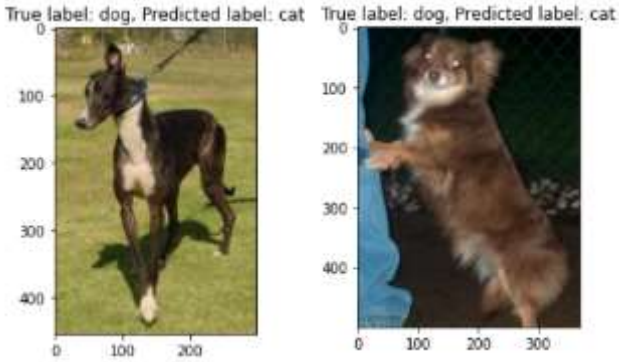


Plot of the accuracy and loss of the model over time.



	precision	recall	f1-score	support
Cat	0.85	0.96	0.90	989
Dog	0.95	0.83	0.89	1011
accuracy			0.89	2000
macro avg	0.90	0.89	0.89	2000
weighted avg	0.90	0.89	0.89	2000

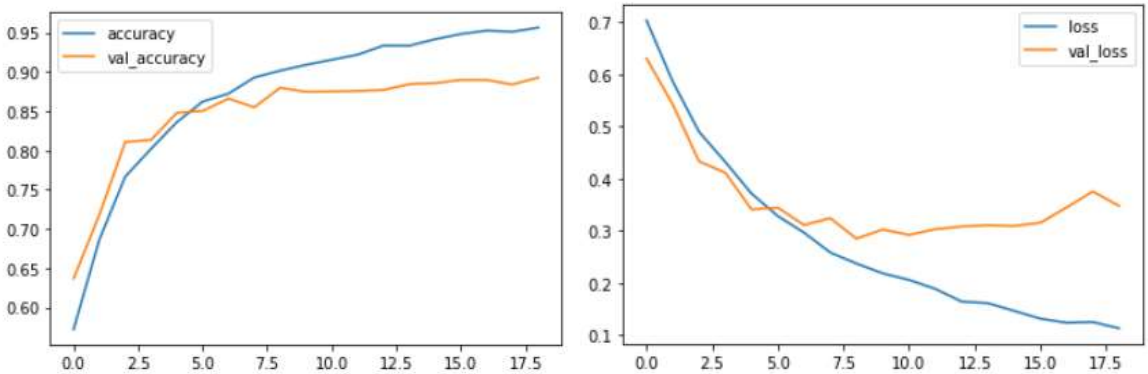
Confusion matrix and classification report of the forth best model.



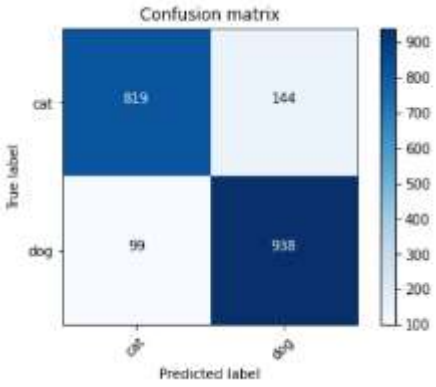
Two misclassified objects

5th best result (Model 1)

Fifth best results were achieved using **model 1**.
Accuracy = 0.8784999847412109.
This is one of the first good models I was able to design, but the results aren't satisfactory.

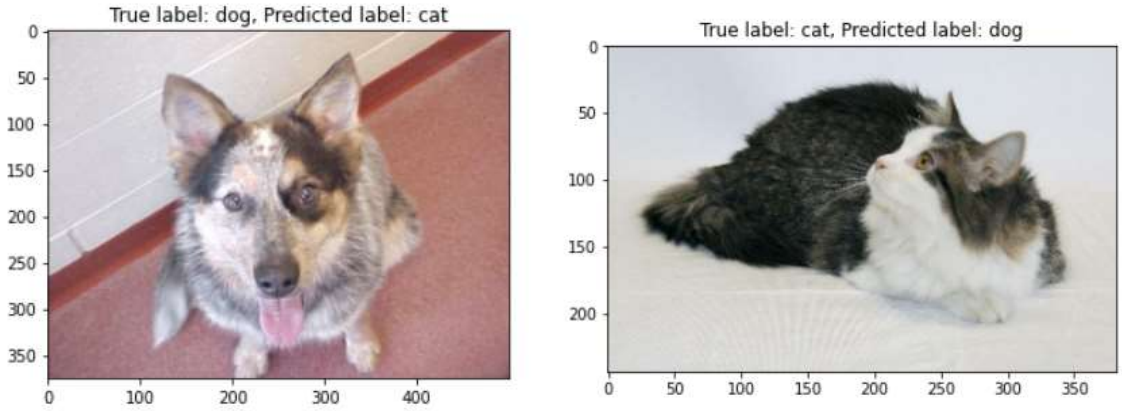


Plot of the accuracy and loss of the model over time.



	precision	recall	f1-score	support
Cat	0.89	0.85	0.87	963
Dog	0.87	0.90	0.89	1037
accuracy			0.88	2000
macro avg	0.88	0.88	0.88	2000
weighted avg	0.88	0.88	0.88	2000

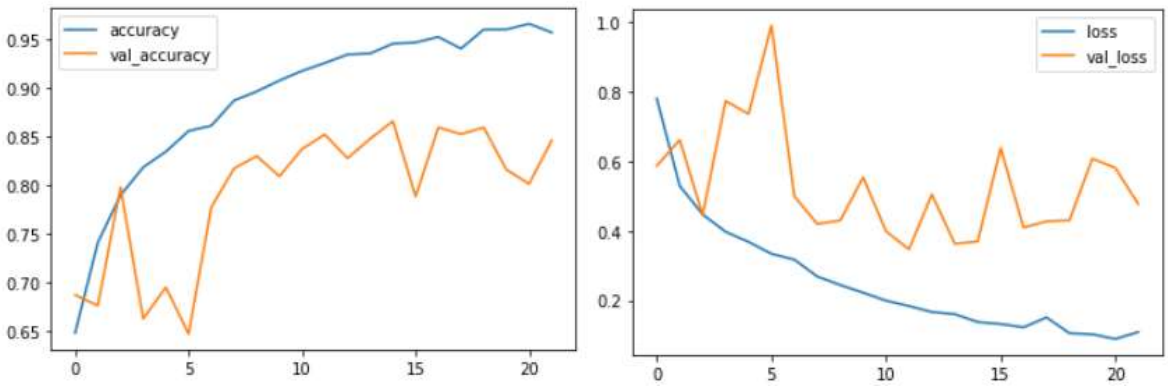
Confusion matrix and classification report of the fifth best model.



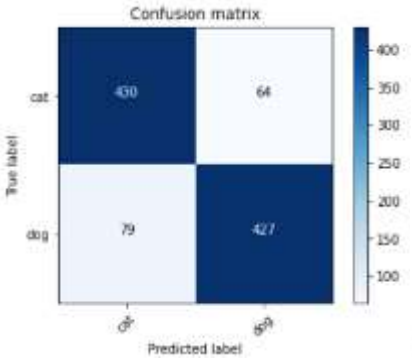
Two misclassified objects

6th best result (Model 3)

Sixth best results were achieved using **model 3**.
Accuracy = 0.8569999933242798.
This is one of the first good models I was able to design, but the results aren't satisfactory.

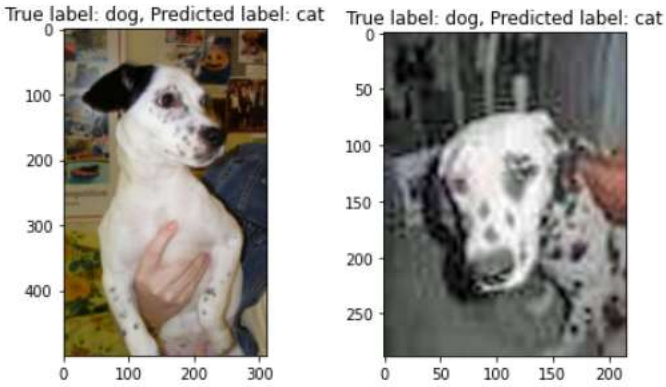


Plot of the accuracy and loss of the model over time.



	precision	recall	f1-score	support
Cat	0.84	0.87	0.86	494
Dog	0.87	0.84	0.86	506
accuracy			0.86	1000
macro avg	0.86	0.86	0.86	1000
weighted avg	0.86	0.86	0.86	1000

Confusion matrix and classification report of the sixth best model.



Two misclassified objects

Discussion.

In summary, the problem addressed in this project is a binary image classification task of classifying images of cats and dogs. A convolutional neural network (CNN) model was used to solve this problem, with the best result being achieved using an EfficientNetB4 model. The model was trained on the "Cats vs Dogs" dataset from Kaggle, and data augmentation techniques were applied to the images to increase the amount of training data available. The accuracy of the model on the test set was ~0.996. Within the project, 6 different models were developed to solve the problem. The loss function used in the project is categorical crossentropy, which is a common loss function used in image classification tasks. In addition to training and testing the model, the performance of the model was evaluated by visualizing its learning curve, confusion matrix, classification report and misclassified objects.

The most challenging part of this project was improving the accuracy of the model. The goal of the project was to train a model that can accurately distinguish between images of dogs and images of cats, and achieving a high accuracy rate was crucial to the success of the project. Several different approaches have been tried to improve the accuracy of the model. Experimented with different architectures and added additional layers to the model, such as convolutional layers and dropout layers. Data augmentation techniques such as rotating, flipping, and zooming were also used to artificially increase the number of training examples, which helped improve model performance.