

CS4287 Neural Computing

Assignment 2: Convolutional Neural Networks

By: Ranya El-Hwigi 18227449 and Nutsa Chichilidze 18131956

Overview





For this project we implemented a Convolutional Neural Network based on the popular CNN architecture Xception. Our CNN works on a cats vs dogs dataset which contains multiple images of cats and dogs and classifies the data accordingly depending on which pet is in the image with an accuracy of 97% which we were very happy with. In this report we go into detail on the dataset, the network and the results we received. We also evaluate our results and explore the impact of varying hyperparameters.

Table of contents

Overview	1
The Dataset	3
The Network: Xception	5
Depthwise Separable Convolution	5
Structure and Architecture	6
Transfer Learning	7
Layers and Hyperparameters	7
Impact of varying hyperparameter(s)	9
Experiment 1: Tuning the dropout rate in the dropout layer	9
Experiment 2: Tuning the activation function of the dense layer	9
Experiment 3: Tuning the number of epochs during training	10
Experiment 4. Tuning the learning rate of the Adam optimizer	10
Results	11
Binary accuracy	11
Binary cross entropy	11
Confusion Matrix	12
Precision	12
Recall (TPR)	13
Area Under the Curve (AUC)	13
Evaluation of the results	14
Binary accuracy	14
Binary cross entropy	14
Confusion matrix	14
Precision	14
Recall (TPR)	15
Area Under the Curve (AUC)	15
References	16

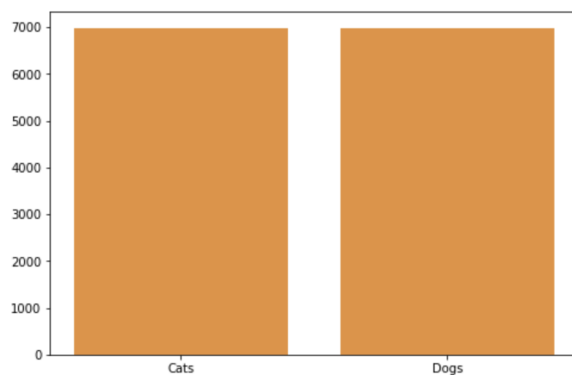
The Dataset

The dataset we chose to use for our project is the Cats vs. Dogs dataset^[4]. It consists of approximately 16,000 various images of cats and dogs.

	image	image/filename	label
0		PetImages/Dog/10396.jpg	1 (dog)
1		PetImages/Dog/4077.jpg	1 (dog)
2		PetImages/Dog/10497.jpg	1 (dog)
3		PetImages/Cat/2763.jpg	0 (cat)

Each sample has 3 features which are image (which is the image itself), image/filename (which is text relating to where to find the image and its name), label (which denotes whether the image is a cat or dog).

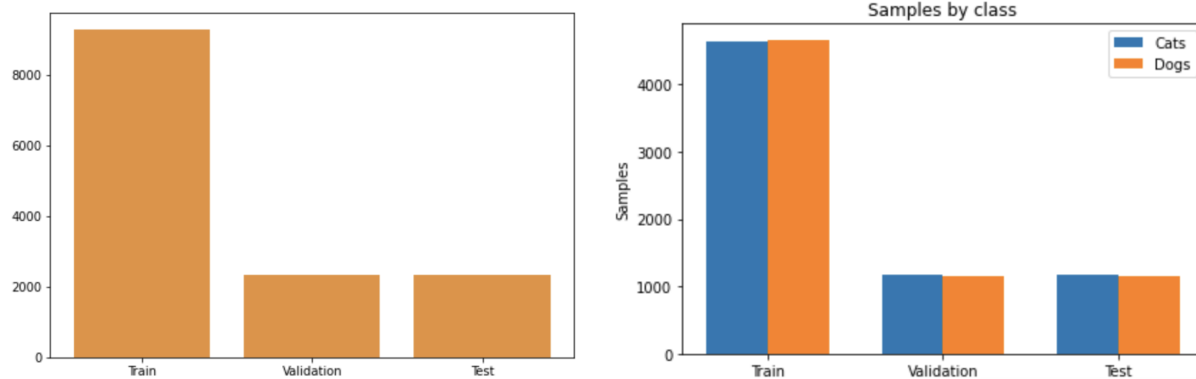
Number of classes, of course, is two (cat, dog) and are denoted with 0 (for dog) and 1 (for cat).



The dataset is fairly split 50/50 with about 7,000 of the images being of cats and 7,000 being of dogs. About 2,000 images are corrupted therefore they're skipped and won't be included in our train, validation, and test sets.

We split the dataset 10% for validation, 10% for test and the remaining for training. This resulted in just over 9,300 samples for training, 2,300 for validation and 2,300 for test. The split of dogs and cats between the sets is approximately equal.

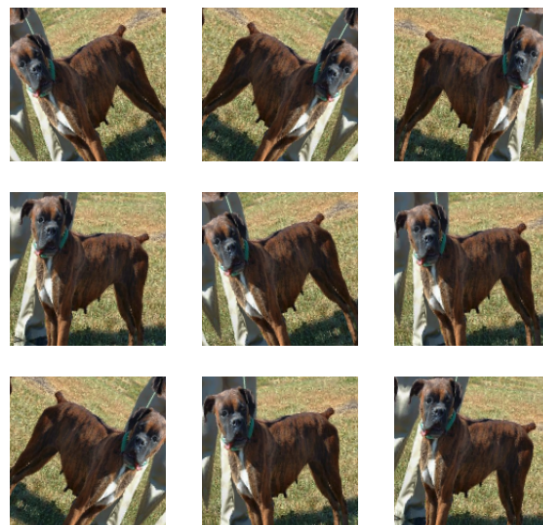
In the figure to the left we can see the total sample split between the train, validation and test sets. In the figure to the right we can see a comparison of the cat samples and dog samples for each set.



Since neural networks receive inputs of the same size we thought it best to resize all the images to the same size square images as we could see some images were portraits and some were landscapes of various sizes. We knew that the larger the fixed size, the less shrinking required and less shrinking means less deformation of features and patterns inside the image.^[3] However, due to the limitation of our computing power we thought it best to stick to a reasonable size of 150x150. For resizing we use tensorflow's `image.resize()` function and mapped it to all three of our sets, train, validation and test.

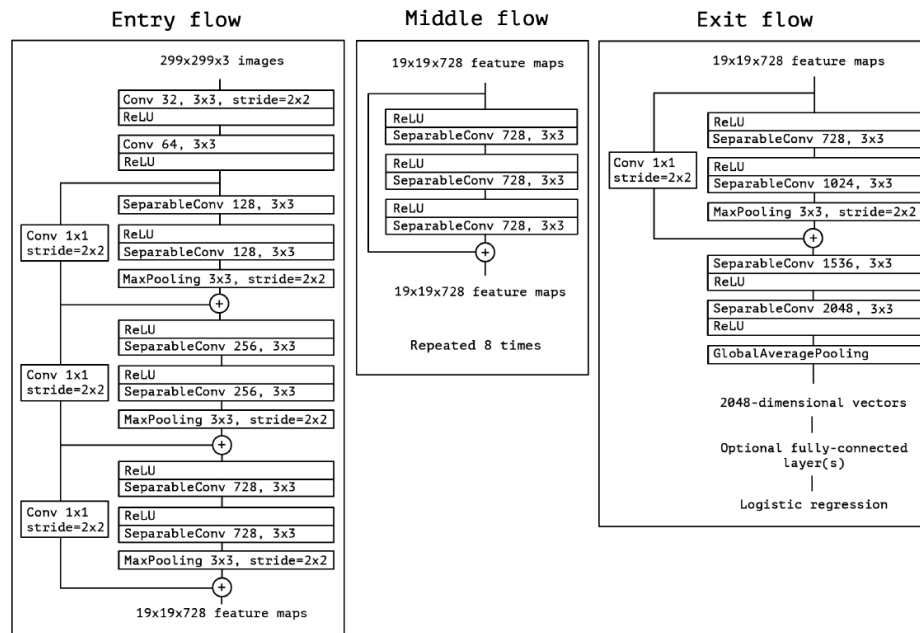
Since the performance of a neural network improves with the amount of data available, we decided to perform data augmentation to artificially create new training data from the training data available. We augmented the data by randomly flipping images horizontally and randomly rotating images by 0.1.

An example of these augmentations performed on an image is given on the right.



The Network: Xception

For our solution we utilized the Xception CNN architecture, which was developed by Google. The name stands for Extreme version of Inception, and is considered a new and improved version of Inception-V3, with modified depthwise separable convolution.



The general network is 77 layers deep and is frequently used for image classification as it has been widely trained on the ImageNet dataset and the pre-trained model is available across multiple frameworks.

The architecture was tested using the ImageNet^[1] dataset, where it outperformed many popular architectures such as VGGNet, ResNet, and Inception-V3.^[2]

Depthwise Separable Convolution

The Xception model is an enhanced version of the Inception model, where the regular modules of the latter were replaced by depth convolutions in Xception.

Depthwise separable convolution is a feature commonly referred to as "separable convolution". It has been a part of the different levels of Inception architecture and has been the main contributor to the speed and efficiency of these CNNs. This type of convolution is readily available to implement using the existing `keras.layers.SeparableConv2d` and `tf.layers.separable_conv2d` layers from Keras and TensorFlow.

The idea behind depthwise separable convolution is that it works with kernels that cannot be split into smaller parts. It deals with both spatial and depth dimensions, i.e. we can have an

image that starts with having 3 channels (RGB color code), and after a few convolutions it might end up with more channels, where each of them interpret different features of this image.

The difference between normal convolution and depthwise separable convolution is efficiency. Normal convolution transforms images multiple times and uses up thousands of multiplications (these calculations can add up to be very costly for a computer, since the multiplication operation is one of the most expensive ones), whereas depthwise separable convolution achieves the same result without actually transforming the image - instead it simply elongates it to work across more channels.

This leaves us with a highly efficient CNN architecture, which has achieved a state-of-the-art status and made Xception one of the most popular CNNs to use today. Logically, the use of depthwise separable convolution has its disadvantages, and can be counter productive for small networks, where the number of parameters are already small. Depthwise separable convolution will attempt to make these parameters even smaller, which in turn might decrease the learning ability of the network.

Structure and Architecture

The base of our model was directly imported and is a built-in keras pre-trained Xception model. The weights and parameters of this model are pre-set by the previous training it went through.

```
base_model = keras.applications.Xception(  
    weights="imagenet",  
    input_shape=(150, 150, 3),  
    include_top=False,  
)
```

The `input_shape=(150,150,3)` matches the rescaling we did in data pre-processing
The `include_top=False` tells the model to not include the ImageNet classifier at the top (the last layer in the neural network, because we will be specifying a new class that we want to do predictions on, which is different from the classes that ImageNet has been trained on.

At the beginning of our implementation, we stop the trainability of this base model, to keep intact the learning it has already done and not to confuse it with new information.

Transfer Learning

Transfer learning allows us to utilize a pre-trained model to calculate predictions for a similar problem. Since the data we were working with was not large enough to achieve training a model with an extremely high accuracy, we opted in to use a pre-trained model which was trained previously on a very large dataset, in our case: ImageNet. (it took Xception 3 days to be fully trained on ImageNet)

The ImageNet dataset is one of the most well-known and frequently used datasets for neural networks/deep learning. It contains millions of images and around 1000 classes out of which we can come across^[1]:

- Maltese dog, Maltese terrier, Maltese, Tibetan terrier, old English sheep dog, German shepherd dog, German police dog, Swiss mountain dog, Bernese mountain dog, French bulldog, coach dog, pug-dog, etc.
- Tabby cat, tiger cat, persian cat, siamese cat, egyptian cat, madagascar cat, etc.

It is safe to assume that the pretrained Xception model has already identified the significant features which can be extracted from images that separate cats from dogs, and with a few added layers and fine tuning, it can work on our dataset of solely dogs and cats.

Keras provides many functionalities to implement transfer learning and fine-tuning. It allowed us to directly import a built-in pre-trained Xception model on the ImageNet dataset.

Layers and Hyperparameters

The use of transfer learning in our solution called for fine tuning when it came to the layers that we put on top of the pre-trained model. This allowed the model to be more fitted to our dataset rather than the dataset it was originally trained on and allowed it to perform with higher accuracy.

- *Normalization layer*: We need this layer to transfer the pre-trained model to our problem with just a binary classification. We use this layer to normalize the previous input of $[0, 255]$ to $[-1, +1]$. Keras provides functionality to directly construct a normalization layer.
- *Global Average Pooling 2D*: This layer takes a tensor (width x height x channels) and counts the number of values across the whole matrix for every channel. It performs the process of merging at the purpose of reducing the size of our data.
- *Dropout*: This layer is a very useful one to avoid overfitting. It randomly picks units to set to 0 with a frequency of our choice. We decided to set the dropout rate to 20% of

weights and trainable parameters for the layer that goes on top of our pre-trained model.

- **Flatten:** This layer is an important part of our convolutional neural network. It serves the purpose of flattening the output of our network into a single vector. It then connects to the final classification model and allows us to classify our data between cats and dogs.
- **Dense:** This layer implements the operation $output = activation(dot\ product(input, kernel) + bias)$. This layer allows us to classify the images in our dataset based on the output from the previous convolutional layers. The dense layer will ensure that the results get classified between a cat or a dog.

Some of the hyper-parameters for the Xception CNN model are:

- **Activation function:** the node at the end or in the middle of neural networks, which introduce non-linearity into the outputs of these neurons.
- **Number of residual blocks :** residual blocks are parts of the network that allow us to train deep neural networks and are used when the activation of a layer is fast-forwarding to another layer into the network.
- **Pooling:** The driving force of the pooling layer, a function that progressively reduces the spatial size of the network, which then makes the network less computation heavy,
- The optimizer and the learning rate of the optimizer
- **Dropout rate:** Dropout is a regularization technique to reduce overfitting (more accuracy on the training data and less accuracy on the test/unseen data). This allows our model to be more robust.
- **Epochs:** the number of times the learning algorithm will work through the entire dataset

We experimented with some of these hyper-parameters in the following section of the report.

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	(None, 150, 150, 3)	0
sequential (Sequential)	(None, 150, 150, 3)	0
normalization_1 (Normalization)	(None, 150, 150, 3)	0
xception (Functional)	(None, 5, 5, 2048)	20861480
global_average_pooling2d_1 (GlobalAveragePooling2D)	(None, 2048)	0
dropout_1 (Dropout)	(None, 2048)	0
dense_1 (Dense)	(None, 1)	2049

```

Total params: 20,863,529
Trainable params: 2,049
Non-trainable params: 20,861,480

```

We can see in the summary of the model that the pre-trained Xception base model provides 20 million parameters, whereas the dogs VS cats data we are working on only provides 2049. The pre-trained model doesn't get trained during transfer learning as it might cause loss of accuracy and completely change the weights of the already "perfect" model.

Impact of varying hyperparameter(s)

During our training process, we identified and documented the impact some of the hyper-parameters might have on the accuracy and general performance of our model. Some of the experiments we carried out were as follows:

Experiment 1: Tuning the dropout rate in the dropout layer

Model	Dropout rate	Binary Accuracy	Loss	Validation binary Accuracy
1	0.2	0.9839	0.0430	0.9798
2	0.4	0.9803	0.0517	0.9794
3	0.6	0.9783	0.0569	0.9802

It was evident that increasing the dropout rate caused a slight increase in loss and a slight decrease in both binary accuracy and validation binary accuracy. This experiment made us decide to stick with the original model with a 20% dropout rate.

Experiment 2: Tuning the activation function of the dense layer

We attempted to add the activation function ReLU to the dense layer as we had read that this function is able to increase the efficiency of a model in a number of cases.

Model	Activation Function on the dense layer	Binary accuracy	Loss	Validation binary accuracy
1	Sigmoid	0.9839	0.0430	0.9798
4	Relu	0.9706	0.3764	0.9819

The decrease in the binary accuracy convinced us that the relu activation function is not suitable for our dense layer and decided to go against using it for training our model. As well as that, the sigmoid activation function allows us to calculate some specific metrics for the model, such as precision and recall.

Experiment 3: Tuning the number of epochs during training

Model	Total number of epochs	Binary Accuracy	Loss	Validation binary accuracy
1	9 (6 + 3)	0.9839	0.0430	0.9798
5	13 (8 + 5)	0.9889	0.0294	0.9811
6	7 (4 + 3)	0.9819	0.0467	0.9841

We tried both lowering and increasing the number of epochs for our model. We saw that the increase in the number of epochs did cause a slight raise in our binary accuracy as well as a drop for our loss score.

Experiment 4. Tuning the learning rate of the Adam optimizer

Model	Learning rate	Binary Accuracy	Loss	Validation binary accuracy
1	1e-5	0.9839	0.0430	0.9798
7	0.0001	0.9754	0.0571	0.9828

Again, due to the drop in binary accuracy and rise in loss, we decided against changing the value of the optimizer learning rate.

Finally, after all our experiments, we found it hard to find a tuning of a hyper-parameter that would give us a better result than the model we originally had.

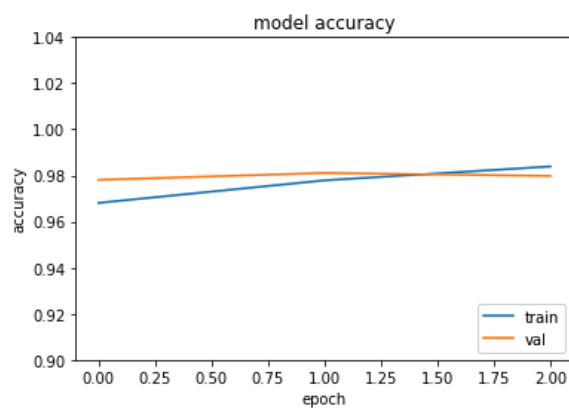
Results

Since we were performing binary classification we decided to go with the following evaluation metric:

Binary accuracy

Binary accuracy calculates how often predictions match binary labels. What this metric does is create two local variables (total and count) which are used to compute the frequency with which the predicted label matches the true label.^[6]

Using `keras.metrics.BinaryAccuracy(name='binary_accuracy')` we got a binary accuracy of 0.97 → 97%.



To the left is a plot of the performance of our binary accuracy over 3 epochs during training. We can see our training and validation accuracy stay close in range throughout showing there is no over fitting.

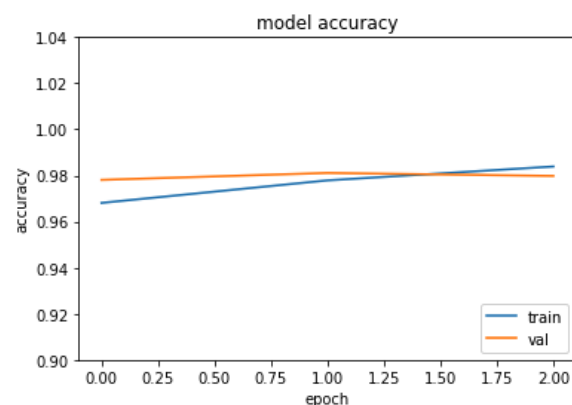
Binary cross entropy

Binary cross entropy computes the cross entropy metric between the labels and the predictions. Cross-entropy is a measure of the difference between two probability distributions for a given random variable or set of events and its widely used as a loss function when optimizing classification models.^[8]

Binary cross entropy compares each of the predicted probabilities to actual class output which can be either 0 or 1 and it then calculates the score that penalises the probabilities based on their distance from the expected value, so how close or far they are from the actual value.^[9]

Using `keras.losses.BinaryCrossentropy(from_logits=True)` We got a binary cross entropy of 0.06.

To the right is a plot of the performance of our binary cross entropy over 3 epochs during training. We can see our training and validation loss stay close in range throughout showing there is no over fitting.



Confusion Matrix

A confusion matrix is a table that is often used to describe the performance of a classification model.^[10]

It's made up of nine values as follows:

T_p : reflect the positive cases that the classifier correctly classified as positive, in our case dogs (label 1)

T_n : reflect the negative cases that the classifier correctly classified as negative, in our case cats (label 0)

F_p : reflect the negative cases (cats) that the classifier incorrectly classified as positive (dogs).

F_n : reflect the positive cases (dogs) that the classifier incorrectly classified as negative (cats).

C_p : reflect the truly positive (dogs) instances in the set.

C_n : reflect the truly negative (cats) instances in the set.

R_p : reflect the number of predicted positive (dogs) instances.

R_n : reflect the number of predicted negative (cats) instances.

N : the total number of samples in the data set.

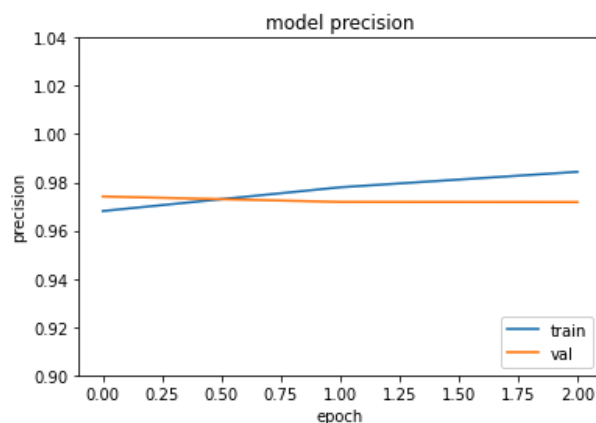
Using `keras.metrics.FalseNegatives()`, `keras.metrics.FalsePositives()`, `keras.metrics.TruePositives()`, `keras.metrics.TrueNegatives()` we obtained the following result:

	Dog	Cat	
Dog	1127	22	1149
Cat	27	1150	1177
	1172	1154	2326

Precision

Precision is the ratio of the number of true positives to the number of predicted positives (true positives plus false positives)^[11]. It's the ability of the classifier to not label positive (dog) a sample which is negative (cat).

Using `keras.metrics.Precision()` we obtained a precision of: 0.96.



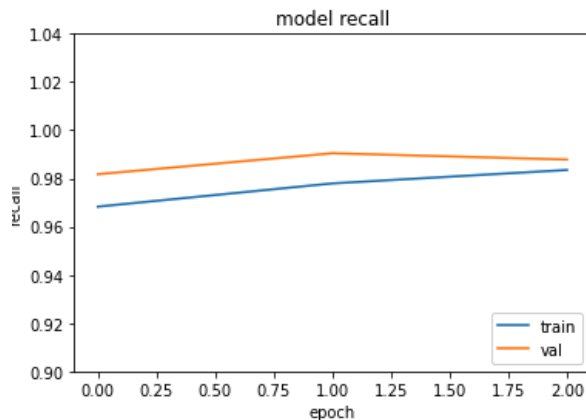
To the left is a plot of the performance of our models precision over 3 epochs during training.

Recall (TPR)

Recall is the ratio of the number of true positives to the number of positives the set contains (true positives plus false negatives)^[12]. It's the ability of the classifier to find all positive (dog) samples.

Using `keras.metrics.Recall()` we obtained a recall of: 0.98.

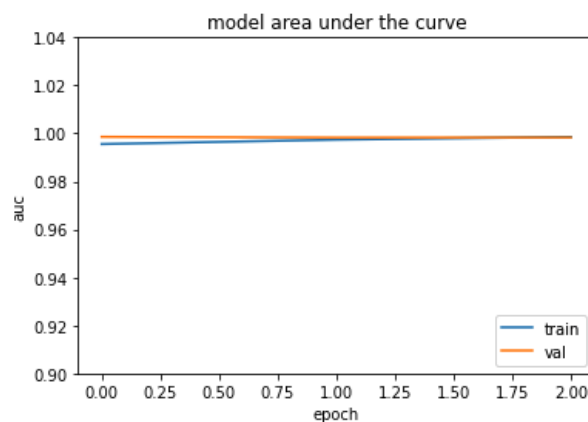
To the right is a plot of the performance of our models recall over 3 epochs during training.



Area Under the Curve (AUC)

AUC-ROC curve is a performance measurement for classifiers at various threshold settings. ROC is a probability curve and AUC represents the degree of separability. It tells us how much the model is capable of distinguishing between classes.^[13]

Using `keras.metrics.AUC()` which default the curve to ROC and the thresholds to 200, we obtained an AUC value of: 0.99



To the left is a plot of the performance of our models area under the curve over 3 epochs during training.

Evaluation of the results

Binary accuracy

With binary accuracy the higher the value we achieve the better with 1 meaning the classifiers predictions are perfect. As mentioned we received an accuracy of 97% so it's fair to say our classifier is highly accurate in classifying dogs and cats, out of 100 samples it would only classify 3 incorrectly.

Binary cross entropy

With binary cross entropy the lower the value we achieve the better with 0 being the perfect loss. As mentioned we receive a cross entropy loss of 0.06 which, accordingly to an article I read on medium^[14], puts in the bracket of "Fine" from the following brackets:

- Cross-Entropy = 0.00: Perfect probabilities.
- Cross-Entropy < 0.02: Great probabilities.
- Cross-Entropy < 0.05: On the right track.
- Cross-Entropy < 0.20: Fine.
- Cross-Entropy > 0.30: Not great.
- Cross-Entropy > 1.00: Terrible.
- Cross-Entropy > 2.00 Something is broken.

However, I think it's fair to point out that we are way below 0.2 and a lot closer to 0.05 so I guess we're on the right track but could definitely improve on our distance from true values.

Confusion matrix

With the confusion matrix what you want to see is low values for false positives and false negatives with 0 for each being the perfect confusion matrix. As mentioned our confusion matrix was as follows:

	Dog	Cat	
Dog	1127	22	1149
Cat	27	1150	1177
	1172	1154	2326

Though not the perfect 0's as we would have liked to see, we were happy enough with our result as we feel a false positive of 22 and a false negative of 27 is fairly low for a model built using google colab.

Precision

With precision the higher the value the better with 1 being the best score and 0 being the worst score. As mentioned we received a score of 0.96 putting us very close to the best score of 1 which tells us that our model is very precise and very unlikely to incorrectly label cats as dogs.

Recall (TPR)

With recall the higher the value the better with the best value being 1 and the worst value being 0. As mentioned we received a recall value of 0.98 putting us very close to the best score of 1 which tells us that our model is very likely to find all positive (dogs) cases in the samples.

Area Under the Curve (AUC)

There are four measurements of AUC:

- $AUC = 1$: tells us this is a perfect classifier and perfect prediction is always obtained.
- $0.5 < AUC < 1$: tells us this classifier performs better than random guess and it has predictive value.
- $AUC = 0.5$: tells us this classifier performs the same as random guess and has no predictive value.
- $AUC < 0.5$: tells us this classifier performs worse than random guess.

As mentioned our AUC evaluated to 0.99 meaning our model performs better than random guess and it has predictive value. (You could even say it's almost perfect).

References

- [1] <https://deeplearning.cms.waikato.ac.nz/user-guide/class-maps/IMAGENET/>
- [2] https://openaccess.thecvf.com/content_cvpr_2017/papers/Chollet_Xception_Deep_Learning_CVPR_2017_paper.pdf
- [3] <https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7>
- [4] https://www.tensorflow.org/datasets/catalog/cats_vs_dogs
- [5] <https://content.iospress.com/download/journal-of-intelligent-and-fuzzy-systems/ifs210925?id=journal-of-intelligent-and-fuzzy-systems%2Fifs210925>
- [6] https://www.tensorflow.org/api_docs/python/tf/keras/metrics/BinaryAccuracy
- [7] https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy
- [8] <https://machinelearningmastery.com/cross-entropy-for-machine-learning/>
- [9] <https://www.analyticsvidhya.com/blog/2021/03/binary-cross-entropy-log-loss-for-binary-classification/>
- [10] <https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/>
- [11] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html
- [12] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html
- [13] <https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5>
- [14] <https://medium.com/swlh/cross-entropy-loss-in-pytorch-c010faf97bab>
- [15] <https://github.com/niconielsen32/NeuralNetworks/blob/main/transferLearning.ipynb>