CMP304 Artificial Intelligence
ML for Computer Vision Project Report [1]

Max Clow - 1904985

[1] Structure, formatting, spelling, grammar, and coherence (5%)

Introduction (5%)

The problem that this report will cover is supervised image classification AI. Image classification is a huge area of AI and has many practical uses (medicine, security, education etc.). The goal of image classification is to train from a data set of images and be able to assign new, unseen images to a certain predefined class. This project deals with Binary image classification, which is only having 2 classes for images to be put into (dog or cat). A multi class image classification system would deal with 2+ classes (bus, car, lorry, van etc,).

This report is specifically dealing with artificial intelligence designed to recognise whether the pictures it receives are of one specific cat, or not (2 classes). The applications of this are relevant, considering the number of smart devices that the average household contains nowadays, smart "cat flap" devices are becoming increasingly more common. Microchip cat flaps have existed for a while however AI can be expanded upon and recognise so much more than just the cat, such as identifying if the cat has any prey it is trying to bring into the house.

The most common and closely related problems that other research has been performed on is the Cats vs Dogs classification. The methods used in these problems are similar, with the dataset going through some level of pre-processing (as most Computer Vision models will do). Once the data is ready, it is put through a Convolutional Neural Network (CNN) in which layers are added to the structure of the model, to help the AI determine patterns in the images and assign a class accordingly. These applications tend to have a high-test accuracy (~90%).

The goal of this project can be applied to many different things. Cat flaps have been mentioned, but it also can be used for photo applications to pre assign pictures of a certain cat to its own folder and for security purposes to identify different cats on a property. The solution for this project is to be a building block that could be implemented into a "smart" cat flap device, or some of the other applications stated above. The solution will make use of many different machine learning libraries (TensorFlow and Keras) and written in Python. The data used will be from my own images, not an existing dataset from online. This data will go through some pre-processing stages for it to be used in the application and ensure it is suitable for the task.

Data Specification (10%)

The data used for this project was custom made specifically for this project. The training/validation data consisted of 200 pictures of one specific cat (obtained through personal photos) and 200 pictures of multiple other cats (obtained from online - https://www.reddit.com/r/cats/). It was important when selecting the data to establish a "baseline" of images, where there was a lot of normal/expected/typical images, including pictures from the front of the cat, the cats face, in common cat poses when lying down etc. Then there was room to put a few more obscure images (blurriness, parts of cat cut off, strange angles, poor lighting etc.) to help the AI try and deal with images that were a bit more difficult to interpret.

These images were all manually cropped to have as much of the image taken up by the cat as possible, to help the AI ignore any backgrounds that might have interfered with the training process. It was especially important to make sure the split between the two different classes of images was even. If there had been 400 other cat images and 100 of specific cat images, the system would have an extreme bias towards the other cat images as it has far more data to work with and learn from. This is not the case for the test data however, as the different metrics for measuring the results do not regard the split between data, they are just interested in how the AI performs. The test data comes in two different formats, cropped and uncropped. The cropped test data consists of 98 images, with roughly a 40/60 split between specific cat and other cat images, respectively. Just like the training data, there are some simple images with good angles, a variety of different poses from cats, obstructions, lighting levels and blurriness. The uncropped test data is the same, with the same images but they have not been cropped, the results of which will be discussed later.

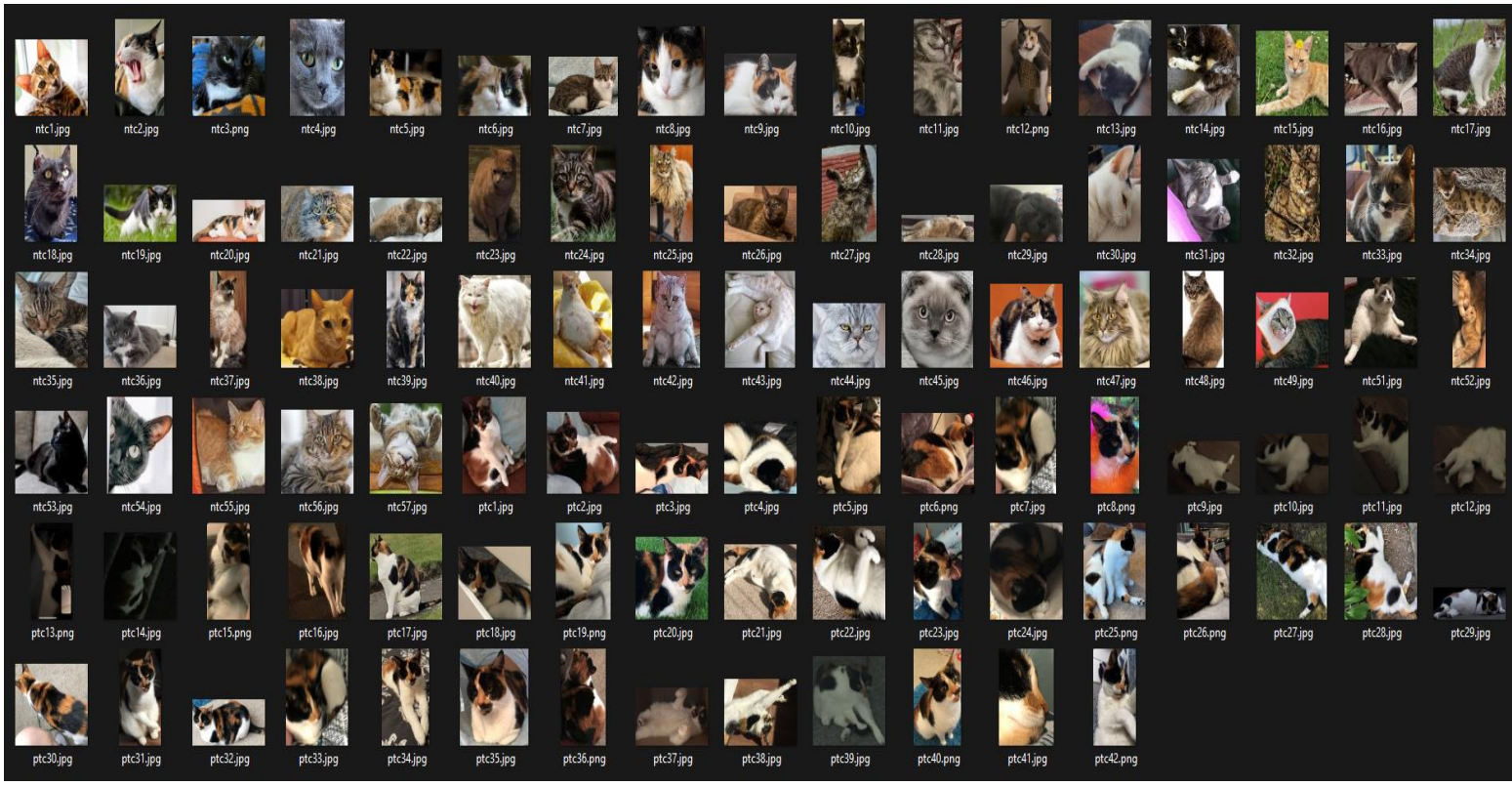
One specific area of interest is the colour of the images. A distinct feature of cats is their colour. If the "one specific" cat was orange, even if there was orange cat images present in the other cat data, the AI would have a tough time determining what class it belongs to. This is particularly true due to bias. If this were the case, the specific cat class would have 100% of its images contain orange, whereas for the other cat's data set, there would be a lot more variation in colours and so there may only be around 10% of those images that include orange colour. This makes the AI become much more bias to the specific cat class when it identifies orange in an image as it has much more reason to assume the image belongs to the specific cat class, when it should not belong to that class. For this reason, all images were grayscaled in the program, to make sure the AI does not develop a bias for colour and can focus on the actual features of the cats. Note that all the test images are in colour and have not been grayscaled.

Ideally, there would be far more images in the dataset for this problem, however, there only exists so many images for the specific cat and as stated earlier, increasing the amount for just the other class would be more detrimental to the project than it would be positive.

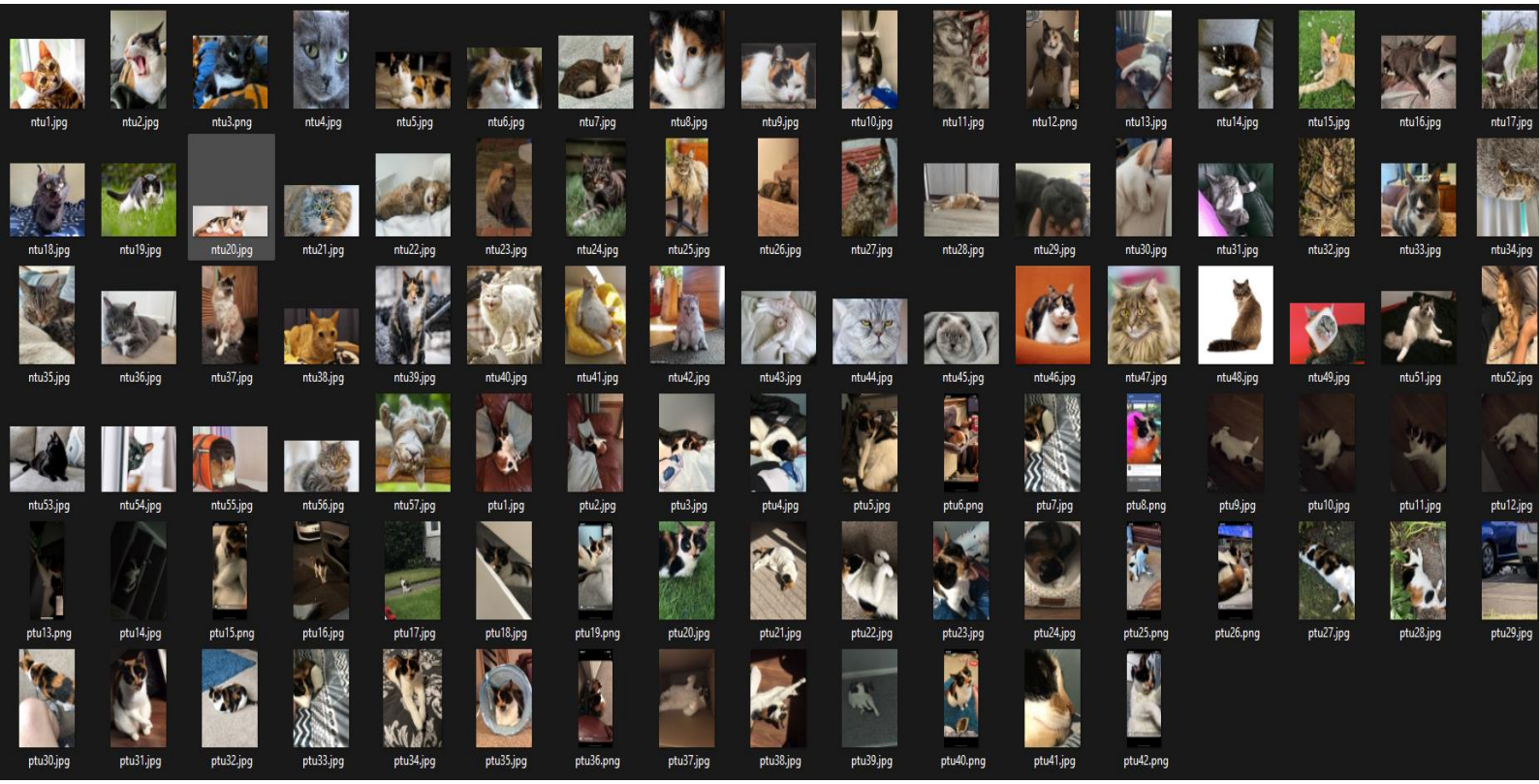
Training - "Tiny" (specific cat class) images

Training - Other cat class images

FULL Test data - Cropped



FULL Test data - Uncropped



Methodology (15%)

Since the project made use of TensorFlow libraries, a new TensorFlow environment had to first be set up in Anaconda which included Jupyter Notebook. Then the libraries were subsequently imported into Jupyter Notebook in the code. From the TensorFlow library, "keras" must be imported as well, as this is the main library used for this application. Also imported was matplotlib and NumPy (for plotting training and validation accuracy), PIL for opening and adjusting images, OS for working with file paths and pathlib to get the path to the images in the correct format for keras.

Data Pre-Processing

As described before, the images were originally in colour and converted to grayscale. This was done by implementing code into the program that is only intended to be run one time. For both the specific cat and other cat folders, these paths must be assigned to variables so that the folders can be iterated over to retrieve each image. Once iterating, there are checks for files ending with .jpg and .png, as these are the only two file types that the data contains. Depending on the filetype, the image is converted to grayscale using PIL and then saved as the appropriate filetype in the correct destination.

Training

Now that the images have been converted to grayscale, there must be a variable that contains the path to the folder with these images. Inside this selected folder, the data must not be "loose" and should be put inside folders with the names of the class they belong to.

Loading the dataset – Now, the data needs to be loaded into the program. At this stage, the batch size, image height and image width should be assigned. Batch size corresponds to the number of training examples in one forward/backward pass. Batch sizes are relevant to the number of epochs in the program, which will be discussed later. The batch size for the data in this project was set to 20. It is important to set a consistent image size for the program to iterate over. Without this step, the program may develop a bias according to the size of the image, or it may learn incorrectly and have a dependence on image size. It also vastly improves performance.

2 variables were created, train_ds and valid_ds. Into these two variables, keras functions are used to load the data from the directory we specified previously, and some parameters are used. Some of the parameters include validation split, which is especially important to make sure there is some data to validate against when training the model. This was set to 0.2, which

allows a decent size of data to compare to. The image size and batch size are also passed into the function. The class names can then be extracted from these variables.

Performance – Some further steps were taken to improve the performance. This included using buffered prefetching which allows production of data from disk in a way that input/output speeds do not become blocking. So, while the model is executing a training step, the next training step is already reading the data. The autotune variable dynamically decides the number of elements to prefetch at runtime. Including cache on top of this keeps the images in memory after they are loaded off disk during the first epoch.

Model – Next, the model needs to be set up. Before that happens, a data augmentation layer should be setup. By doing this, the chance of overfitting is drastically reduced. Overfitting refers to the model fitting the data so well to the point where it cannot accurately predict new, unseen data, which should be avoided at all costs, as it defeats the purpose. To make the model more stable due to the smaller dataset size, data augmentation is applied. This makes use of keras functions and applies layers to the model that will randomly flip (horizontally or vertically), rotate, or zoom on the images. This layer will then shortly be applied to the model.

A new variable is setup (named model). This should be set equal to a "Sequential" function, to set up the model. This function takes the parameters of what layers that the model should contain. A CNN combines learned features with input data and does so using 2D layers (ideal for 2D images that this project uses). There are 3 levels of layers in a CNN, an input layer, hidden layers, and output layer. Hidden layers can contain a large variety of types of layers. The augmentation layer that was set up previously should be the first layer defined in the model. Next, there should be a normalisation layer. This is because while the images will have RGB values up to 255, which is not ideal for a neural network as the input values should be as small as possible. This normalisation layer caps the RGB values to be between 0 and 1. Next ReLU layers should be applied to increase non-linearity, with a pooling layer added on top of each one. For this project, 3 ReLU layers were applied, with sizes from 8 to 32. After this, the pooled images should all be flattened. Flattening converts all the pooled images into one vector. After this, 2 Dense layers are applied. One with a ReLU activation and the other with a softmax (which converts a vector to a probability, which will be used to predict a class probability for the 2 classes in the dataset). In-between these a dropout layer should be applied. Dropout reduces overfitting, just like data augmentation, and does so by removing a selection of a certain number of units in a layer. For this project, the dropout value was set to 0.4.

Compiling the model – Now that the model and its layers are set up, the model needs to be compiled. The compile function takes 3 important values. First, is the optimizer type. This can be chosen manually and allows the user to set a learning rate for the model. This project uses the predefined 'Adam' optimizer. The loss part of the function for this project is set to

SparseCategoricalCrossentropy(). Typically, this loss includes from_logits as a parameter, but as the model has already been normalised via the Dense Softmax layer, it is not needed. The final value to pass into the compile function is metrics. Accuracy is a metric measured for the project, so this is set accordingly. At this point a summary of the model can be retrieved

Running the model – Now it is time to run the model. This is where the number of epochs is set. An epoch is equal to one "forward pass" AND a "backward pass". To explain, this project contains 320 training examples (400 total –20% for validation) and the batch size is set to 20. This means it will take 16 iterations to complete one epoch. An epoch determines how many times the model will iterate over the data. After multiple tests, an epoch size of 20 was where the accuracy numbers tended to plateau, hence a size of 20 was incorporated. Now, a fit function should be run on the model, passing through the train_ds, valid_ds and the number of epochs. When this code runs, the model will be learning from the data.

Plotting model training/validation accuracy (optional) - If dealing with a different dataset, you may want to analyse the accuracy of your training before moving onto test data. This is where matplot is used and the Training Accuracy, Validation Accuracy are plotted onto a graph, as well as Training Loss and Validation Loss. These graphs will help determine whether underfitting/overfitting is occurring on the data.

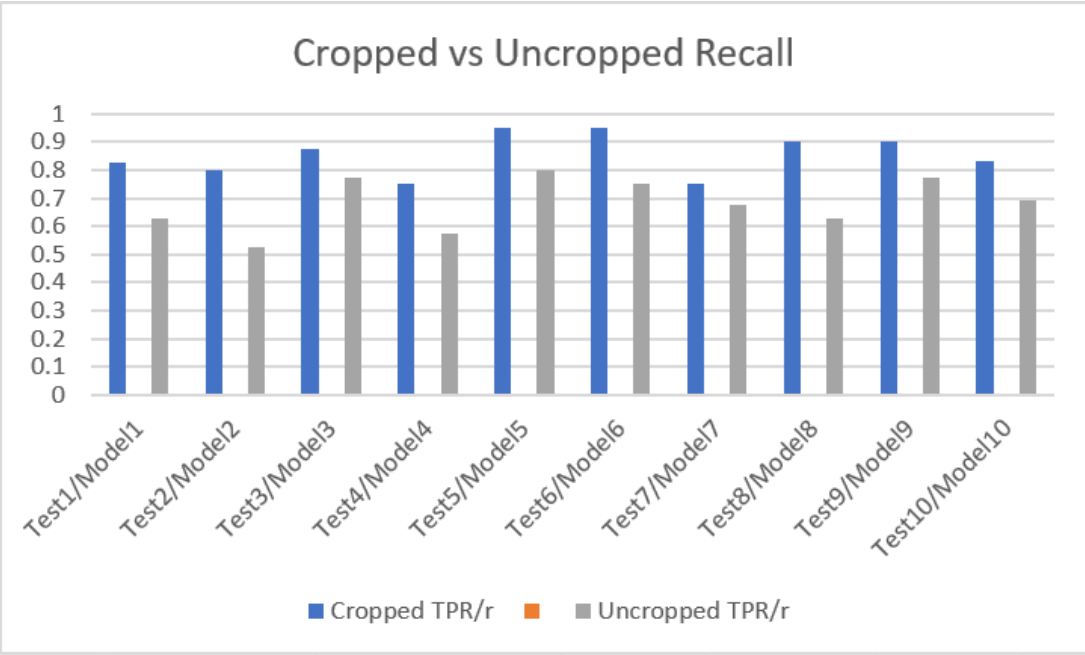
Testing the model – Finally, the model needs to be tested to understand how good it is at predicting the class of a new image. As mentioned previously, the test data is its own data set of 98 images. Here, we want to go to that file path and iterate over the images. For each iteration, the image is loaded to the model which can then be predicted using the predict function to determine what class the image belong to, as well as providing a score of how "confident" the AI is in its choice. Then the image, prediction and score should be output. For the purposes of this project, this step is done twice. Once for Cropped test data and one for uncropped.

Results (10%)

As seen previously in the report, the test data does differ from the training data. While the train data does contain some obscure images, the vast majority are normal and clear. In the test data, there is an abundance of strange positions, blurriness, obstructions/cut-offs of the cats, poor lighting etc. This ensures that the tests are non-biased and does not allow the model to easily fit the test data to a training image it has learned from and ensures rigorous and proper testing.

The tests for this project looked at the difference in metrics for test data that had been cropped, and the same test data that had not been cropped. For both, 10 models were created and trained and then the test data applied to. This ensured that the numbers are reliable and eliminates any extreme (poor or exceptional) results, and an average was obtained. In the tables below, direct comparisons between cropped and uncropped for each type of metric can be viewed:

Recall



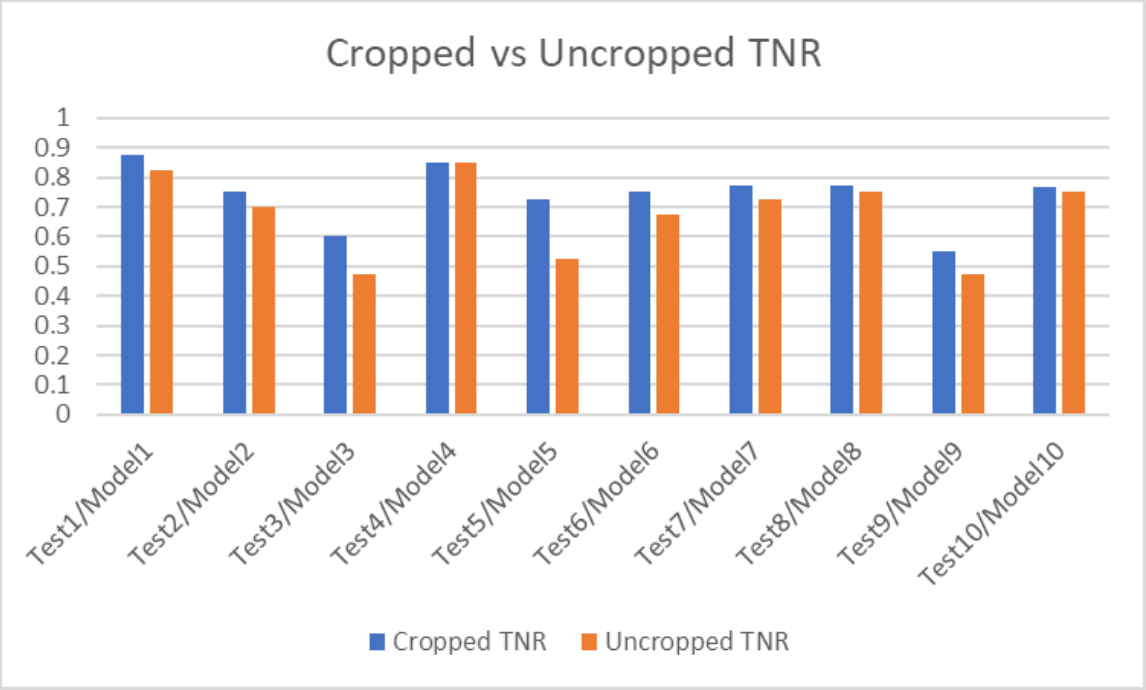
The recall rate (or True Positive Rate) is shown above. As will be a common theme throughout, the cropped recall rate is significantly higher than the uncropped rate. A strong recall rate indicates how good the AI is at predicting correct "positive" results. In this projects case, a positive result is an image of the specific cat, "Tiny" and a negative is another cat. The averages calculated for both were:

Cropped – Recall rate = 0.8533

Uncropped – Recall rate = 0.6815

This is a strong performance from the application for recall rate

True Negative Rate



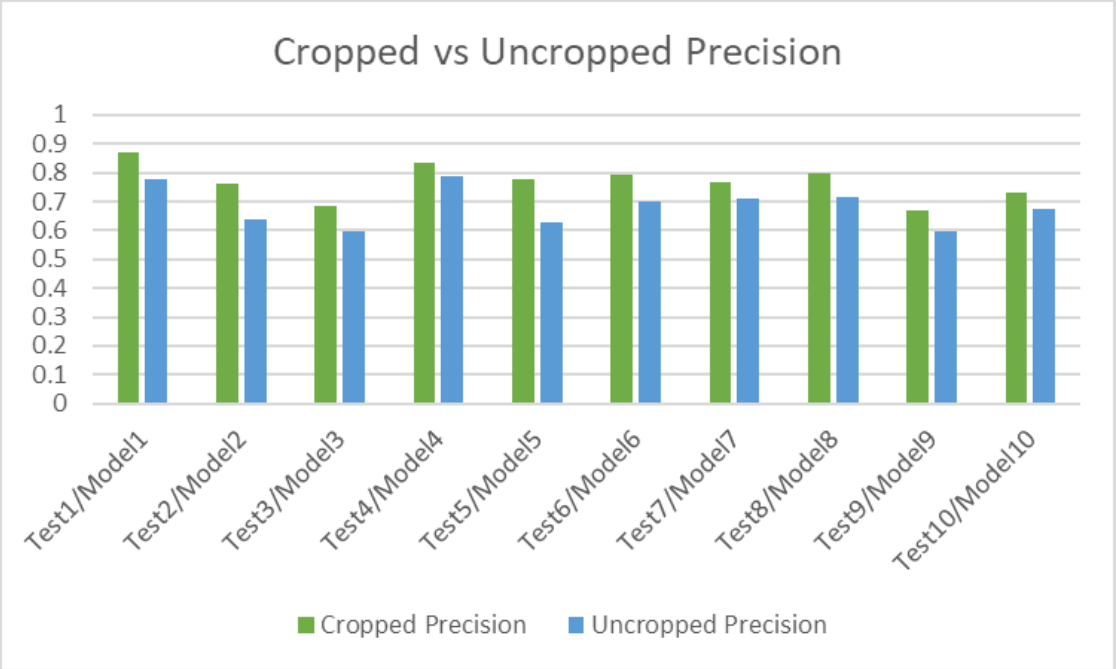
The True Negative Rate is like the recall rate, but instead shows how good the AI is at predicting correct "negative" results. As can be seen above, the difference between cropped and uncropped is not as large as the recall rate difference was. This shows that when it comes to identifying negative results correctly, the impact of the image not being cropped is not as large as it is for positive results, however there is still a clear benefit to cropping. Again, the rate is still high, which means the AI is good at predicting correct negative results. The averages computed are below:

Cropped – Average TNR rate = 0.7418

Uncropped – Average TNR rate = 0.675

Compared to the recall rate, the Cropped TNR rate is quite a bit lower. This shows that the application dealt with "Tiny" pictures well, but not other cat pictures.

Precision



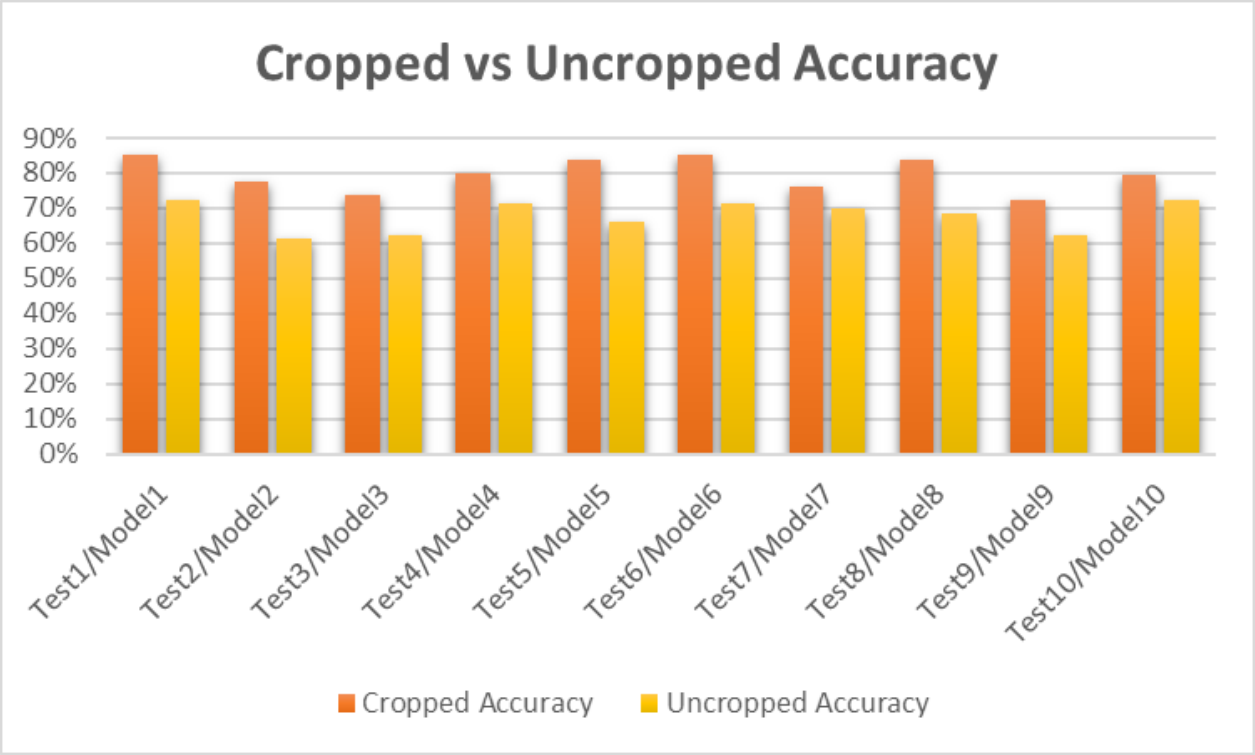
Precision shows how accurate/reliable the model is when it says that a test image is positive. Unlike recall, this number can be affected by False Positive results, e.g. If there are samples of other cat images that the AI determines is a positive result, this will lower the precision. There is still a difference between cropped and uncropped precision rates, with cropped giving higher values again. Averages are:

Cropped – Precision = 0.7683

Uncropped – Precision = 0.6822

This is a good performance from the application for precision

Accuracy

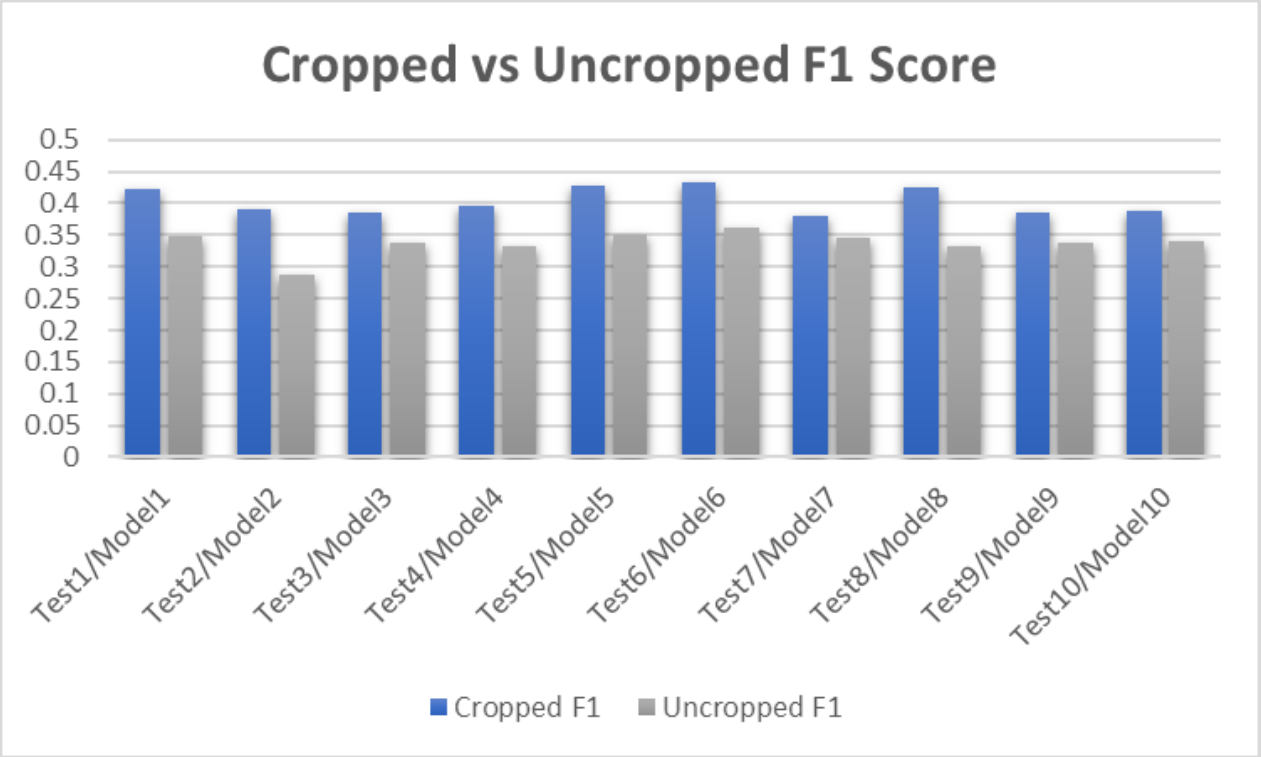


Accuracy is the simplest metric to show AI performance, however, it does not work well with an unbalanced set of test data, like this project. Still, this graph goes to further demonstrate the higher values given by a cropped test vs an uncropped test. The averages computed were:

Cropped – Accuracy = 80%

Uncropped – Accuracy = 67.87%

F1 Score



The F1 score can make up for the shortcomings of accuracy stated above. It is still a measure of accuracy but is suitable for use when the data set is unbalanced. It combines the precision and recall of the model and normally defined as the ”harmonic mean” of the model's precision and recall. As seen above, the cropped data yet again has a higher value than uncropped. The averages are again below:

Cropped – F1 score = 0.4031

Uncropped – F1 score = 0.3376

Discussion/Conclusion (10%)

The goal of the project was to create a Binary image classification AI to identify whether a received picture of a cat was a specific cat, named Tiny, or another cat. This was accomplished, although there is still room for improvement. From the results obtained in the previous section, it can be surmised that the AI did a decent job at determining the class of the image. A precision and recall between 0.75 and 0.85 for a complex, real world problem is very respectable. The reason this was not higher, is due to the lack of training data available. With only 200 images for each class, it is limiting the AI in how much it can learn. With a larger data size (say around 1000 images for each class), it is expected results would be vastly superior.

For this project, it is important that the accuracy measurements are not investigated too much (although they look like strong numbers). In a scenario where the AI is determining if the correct cat is present, precision should be considered more important than recall. The

application would not want to classify an incorrect cat as a positive result and grant that cat entry through a cat flap. Recall focuses on purely positive results and has no interest in what true/false negatives were determined, which is not a key factor that contributes to the goals of the application. If recall were also important to the project, the F1 score would be ideal in measuring performance of the application. It is also worth noting that the test data set was large, allowing for more reliable results, as well as the fact 10 tests for obtaining metrics were performed.

With Cat vs Dog classification applications, "it is routine to achieve approximately 80% accuracy with a manually designed convolutional neural network" (Machine Learning Mastery, 2019). This is a similar number to what this project achieved. In practice, most will achieve higher, especially ones with a larger dataset. This is due to the distinction between cats and dogs being far easier to determine compared to an interspecies comparison trying to identify just one instance of that species vs the rest of the species.

The model and application are well constructed. The performance is high, although this could very well be affected by the smaller dataset. If the dataset were to increase, the model training time would take significantly longer, although precautions were still taken to combat this such as the buffer and pre-caching. The layers implemented into the model are exactly what would be expected to solve a problem of this kind and is not the reason for any shortcomings. Gray scaling the images for training was a clever idea to allow the AI to train more on the salient features of the cats rather than depending/focusing on colour.

To build on top of this application and implement into a working "smart" cat flap, it would be implemented into a device with a camera attached and when motion was detected, it would start taking images over a set amount of time (a few seconds) and determine whether these images were of the cat belonging to the household. If yes, it would unlock the mechanism to allow the cat to come through the doorway. Of course, the user would have to first provide the device with pictures of the cat (through an app, which would then transfer the data over Bluetooth). Other cat images would be preloaded into the device. To improve the device even further, some unsupervised learning could be incorporated so that on top of the original data received, it could train/learn from correct predictions it has made previously.

To conclude, the application was built to determine if images entered are of a specific cat or not and did so with decent results. The CNN built is strong and is not the limiting factor of the application. It accommodates for overfitting and has good precision, an important metric of the system. The data pre-processing was also executed well and helped to improve the system even more. To increase the quality of the application, the dataset would have to be expanded upon to allow the AI to train from a larger dataset.

References (5%)

Label Flow (no date) *The Ultimate Guide to AI Metrics and Evaluation- Image Classification.* Available at: https://labelflow.ai/posts/ai-metrics-image-classification (Accessed: 5 May 2022)

TensorFlow (no date) *Basic Classification.* Available at: - https://www.tensorflow.org/tutorials/keras/classification (Accessed: 29 April 2022)

Analytics Vidhya (2020) *Create your own image classification model using Python and Keras.* Available at: https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/ (Accessed: 1 May 2022)

TensorFlow (no date) *Load and pre-process images.* Available at: https://www.tensorflow.org/tutorials/load_data/images (Accessed: 29 April 2022)

TensorFlow (no date) *Image Classification.* Available at: https://www.tensorflow.org/tutorials/images/classification (Accessed: 28 April 2022)

StackOverflow (2018) *Difference between accuracy and validation accuracy.* Available at: https://stackoverflow.com/questions/51344839/what-is-the-difference-between-the-terms-accuracy-and-validation-accuracy (Accessed: 3 May 2022)

Towards Data Science (2019) *The Complete Learners Guide to Deep Learning.* Available at: https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb (Accessed: 1 May 2022)

Stack Exchange (2017) *Why is the validation accuracy fluctuating?* Available at: https://stats.stackexchange.com/questions/255105/why-is-the-validation-accuracy-fluctuating (Accessed: 2 May 2022)

StackOverflow (2019) *Overfitting on Image Classification.* Available at: https://stackoverflow.com/questions/59201907/overfitting-on-image-classification (Accessed: 2 May 2022)

RoboFlow (2020) *Train, Validation, Test Split.* Available at: https://blog.roboflow.com/train-test-split/ (Accessed: 28 April 2022)

Medium (2021) *Data Augmentation using Keras Preprocessing Layers.* Available at: Data Augmentation using Keras Preprocessing Layers. | by Fabian Christopher | featurepreneur | Medium (Accessed: 1 May 2022)

Stack Exchange (2015) *What is batch size in Neural Network.* Available at: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network (Accessed: 28 April 2022)

Stack Overflow (2017) *Running TensorFlow in Jupyter Notebook.* Available at: https://stackoverflow.com/questions/43216256/running-tensorflow-in-jupyter-notebook (Accessed: 28 April 2022)

UpGrad (2021) *Image Classification in CNN.* Available at: Image Classification in CNN: Everything You Need to Know | upGrad blog (Accessed: 1 May 2022)

Machine Learning Mastery (2019) *How to classify photos of dogs and cats.* Available at: https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

Towards Data Science (2020) *CNN Image classification: Cat or Dog.* Available at: https://towardsdatascience.com/cnn-classification-a-cat-or-a-dog-568e6a135602 (Accessed: 2 May 2022)

Medium (2018) *Image Classified – Cats vs Dogs.* Available at: https://gsurma.medium.com/image-classifier-cats-vs-dogs-with-convolutional-neural-networks-cnns-and-google-colabs-4e9af21ae7a8 (Accessed: 2 May 2022)

Analytics Vidhya (2021) *Beginner friendly Project- Cat and dog classification using cnn.* Available at: https://www.analyticsvidhya.com/blog/2021/06/beginner-friendly-project-cat-and-dog-classification-using-cnn/ (Accessed: 2 May 2022)

Google Colab (2018) *Cat vs Dog Image Classification.* Available at: https://colab.research.google.com/github/google/eng-edu/blob/main/ml/pc/exercises/image_classification_part2.ipynb?utm_source=practicum-IC&utm_campaign=colab-external&utm_medium=referral&hl=en&utm_content=imageexercise2-colab#scrollTo=ijUDyVZtSgz3 (Accessed: 1 May 2022)

The Verge (2019) *Amazon employee made an AI-Powered cat flap.* Available at: https://www.theverge.com/tldr/2019/6/30/19102430/amazon-engineer-ai-powered-catflap-prey-ben-hamm (Accessed: 4 May 2022)