SYSTEMS WITH MACHINE LEARNING

CLASSIFICATION OF ANIMALS
REPORT 4

Adam Cherek, index 207250
Mateusz Nowak, index 189420
Paweł Mańczak, index 188756

We hereby declare that no artificial intelligence (large language models, ChatGPT, etc.) was used to create this report, its text, drawing conclusions, analyzing results and other aspects.

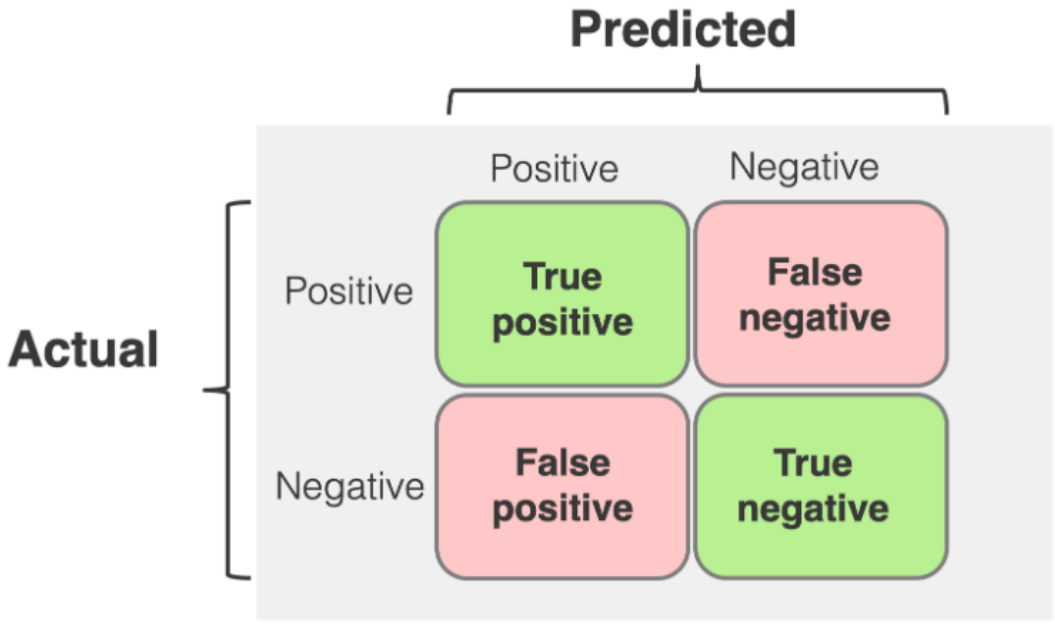
……Adam Cherek… …….Mateusz Nowak…… ……Paweł Mańczak……..

Available Testing Metrics for Multi-Class Classification

Before we introduce metrics it is important to say something about confusion matrix, because other metrics are based on that.

Confusion matrix

A confusion matrix is a fundamental tool for evaluating classification model performance in machine learning. It's a table that provides detailed information about how well a model predicts different classes by comparing actual labels with predicted labels. For binary classification, the confusion matrix has 2×2 dimensions where columns represent actual classes and rows represent predicted classes. The main elements include: True Positive (TP) - correctly predicted positive cases, True Negative (TN) - correctly predicted negative cases, False Positive (FP) - incorrectly predicted as positive (Type I error), and False Negative (FN) - incorrectly predicted as negative (Type II error).



https://www.evidentlyai.com/classification-metrics/confusion-matrix

Accuracy

Accuracy is the most fundamental metric, measuring the proportion of correctly classified images across all classes. It's calculated as:

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

and provides an overall performance indicator. However, accuracy can be misleading with imbalanced datasets where it might reflect the predominance of the majority class rather than true model capability

Precision

Precision measures the proportion of true positive predictions among all positive predictions for each class. For multi-class problems like animal classification, precision is

calculated class-wise, showing how often the model correctly identifies each animal type when it predicts that class. This metric is crucial when false positives are costly - for instance, incorrectly classifying if someone in a photo is a thief.

Recall (Sensitivity)

Represents the proportion of actual positive instances that were correctly identified by the model.

$$Recall = \frac{TP}{TP + FN}$$

In the context of animal classification, recall indicates how well the model finds all instances of each animal type in the dataset. High recall means the model successfully captures most animals of a specific class.

F1 Score

F1 Score is the harmonic mean of precision and recall.

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

The F1 score is particularly valuable because it emphasizes the effect of the smaller value, ensuring both precision and recall contribute equally to the assessment

Why Accuracy

Accuracy is suitable for our project because we have a balanced dataset with 800-1200 images per category (cats, dogs, horses, humans). Since all classes have similar amounts of data, accuracy gives us meaningful results without being biased toward any particular animal type. Our model achieved 71% accuracy, which is much better than the 25% random guess for four classes.

Why F1 Score

F1 Score balances precision and recall, which is crucial for animal classification. We need both:

Precision: When we say it's a cat, we want to be right most of the time

Recall: We want to catch most of the cats in our dataset

F1 Score prevents the model from being too conservative (high precision, low recall) or too liberal (low precision, high recall). This balance is especially important when distinguishing between similar animals like dogs and cats, where mistakes affect both classes.

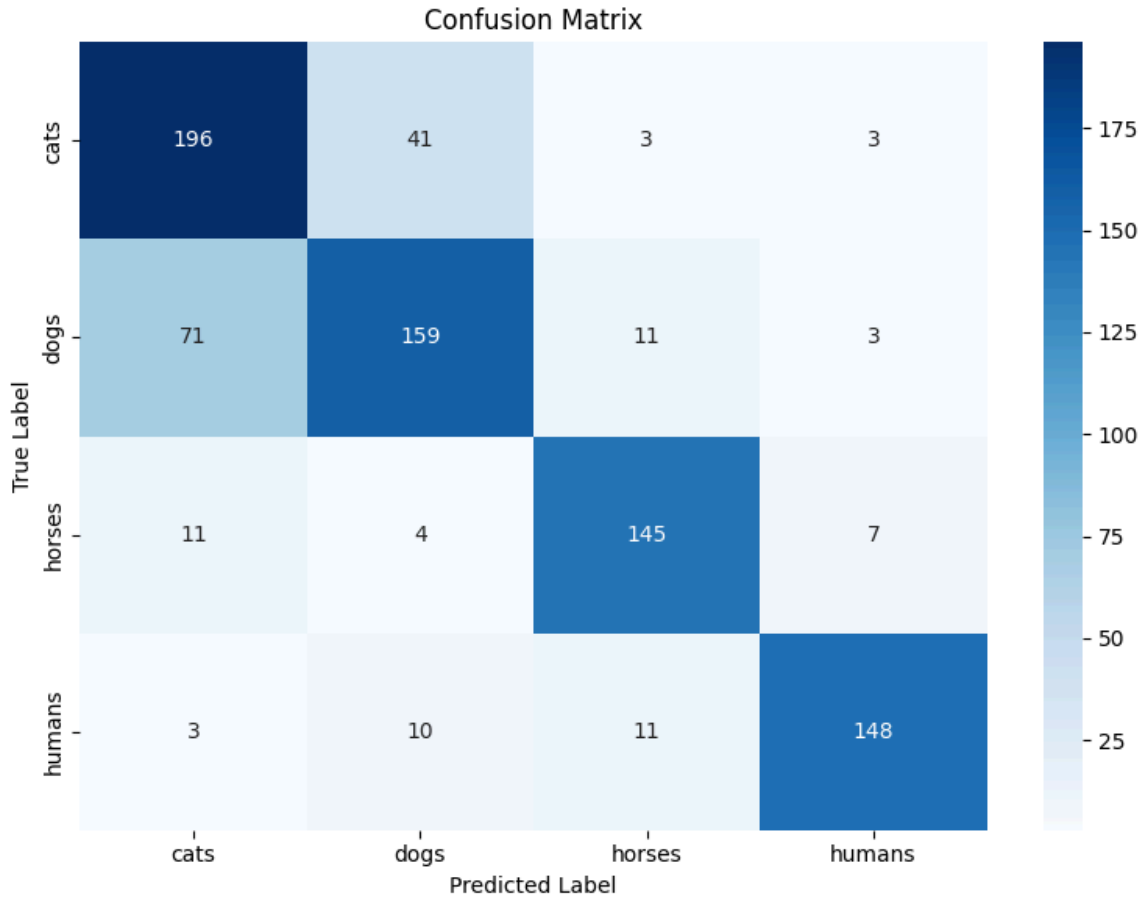
Confusion matrix

We repeated the training and testing on Split2. We came up with classification report for all of the classes. The results are shown below:

	precision	recall	f1-score
cats	0,6975	0,8066	0,7481
dogs	0,7430	0,6516	0,6943
horses	0,8529	0,8683	0,8605
human	0,9193	0,8605	0,8889

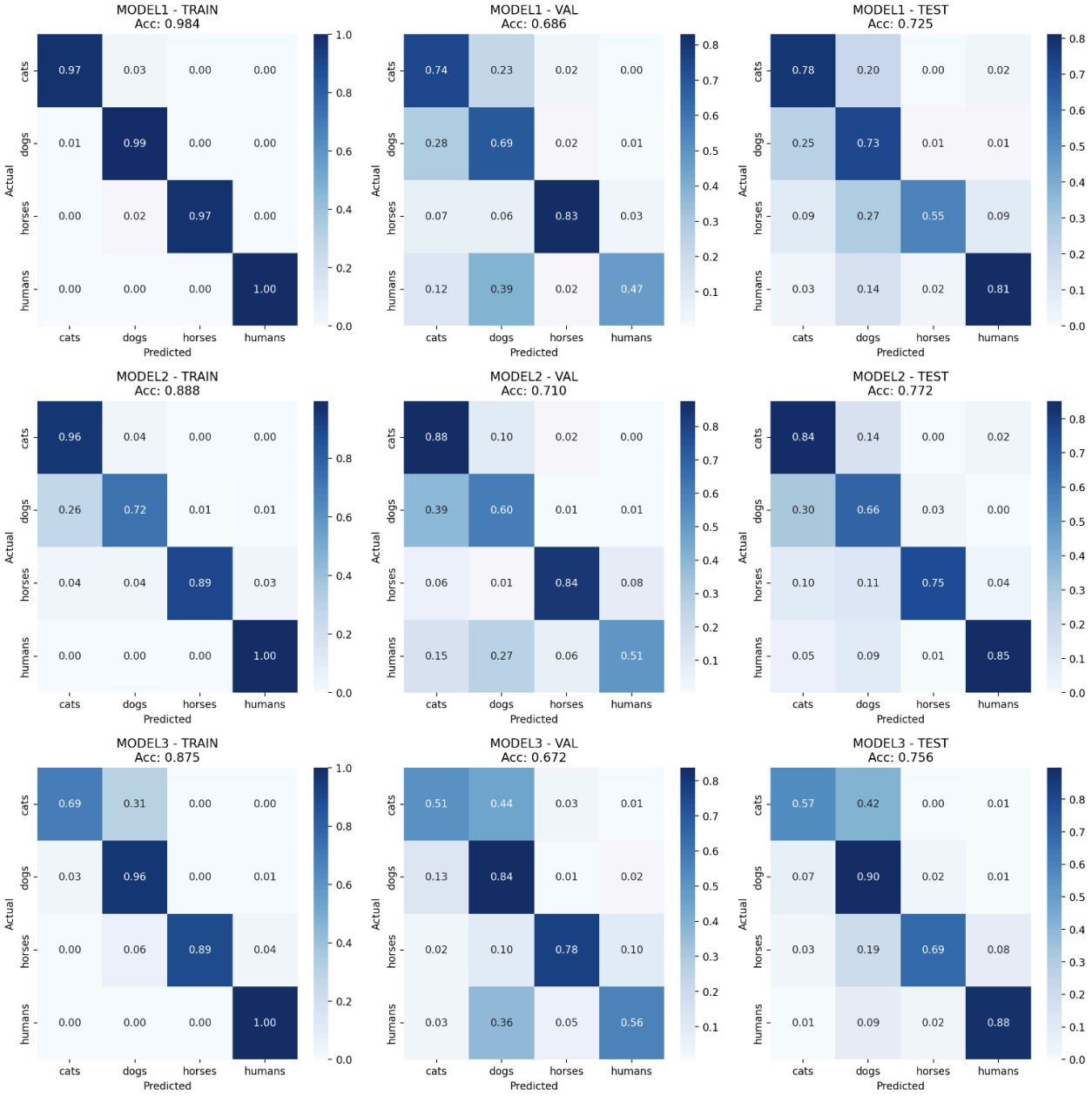
We can see from this data that horses and humans were classified vary good way above the 75% target. On the other hand cats and dogs came out poorly. From our interpretation we think that they were mistaken for each other because of similarities in their appearance. Both are small furry animals. Very interesting number is the recall on dogs which means that it was the class that gave false negative answer a lot of times, once in every three test images.

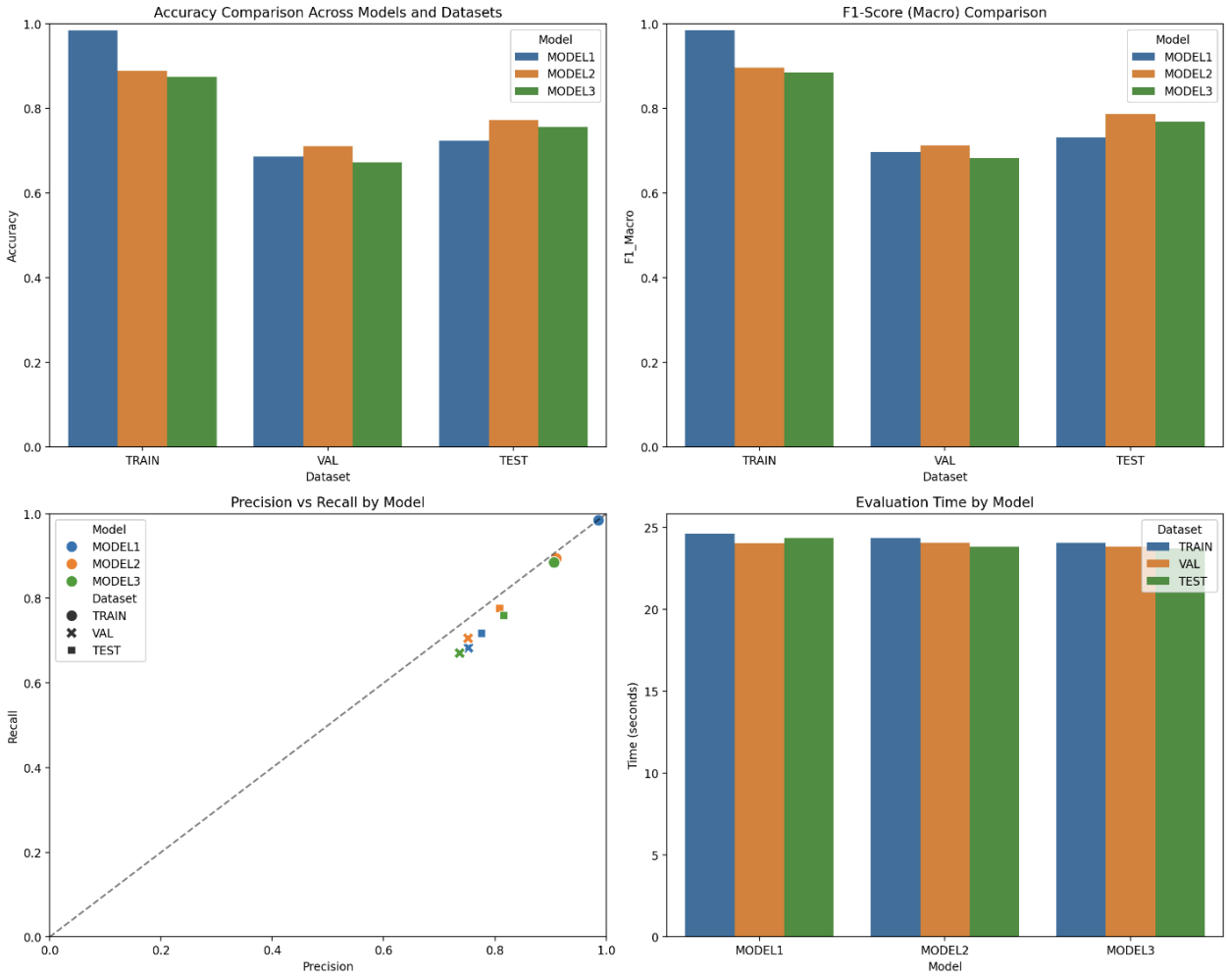
We made a confusion matrix which even clearer shows the problem with cats and dogs classification

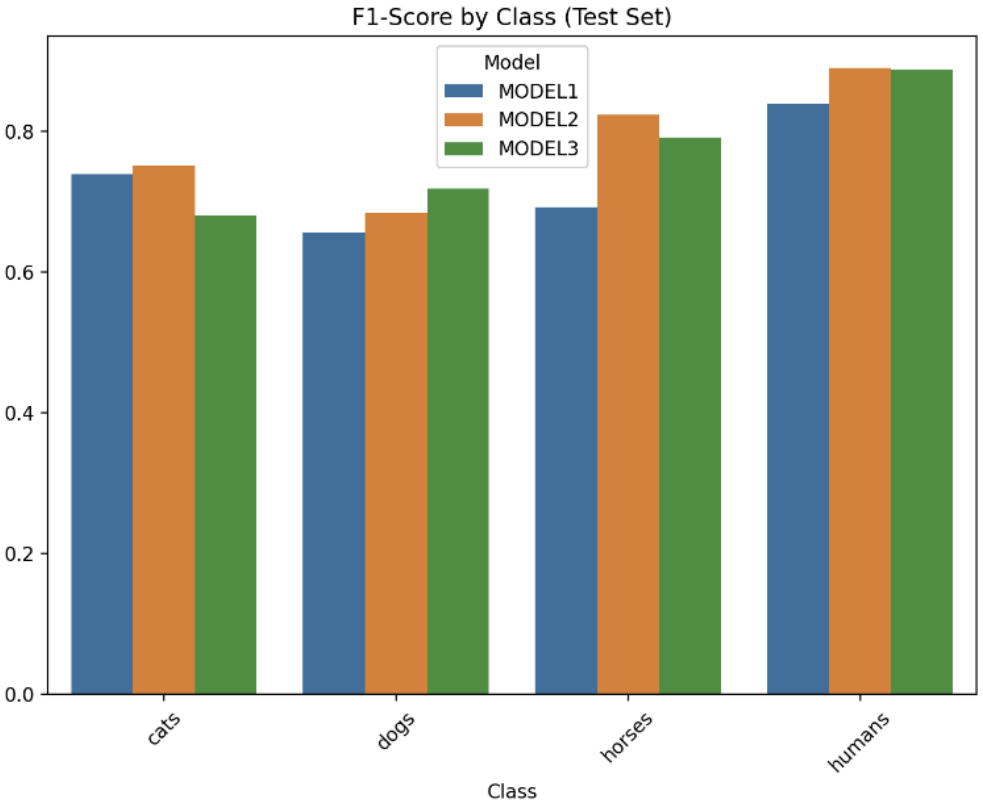


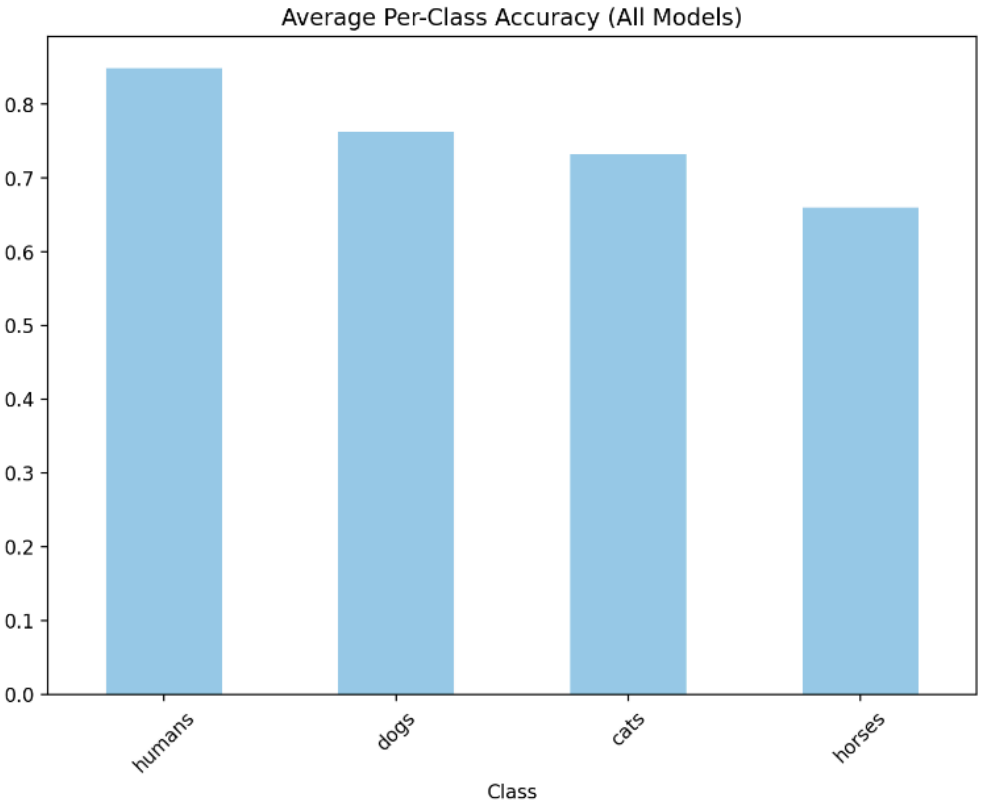
As we can see our interpretation was right. Most of the time dogs were miss labeled as cats. The other way around occurred almost twice as fewer times.

Testing models


MODEL1 - TRAIN
Acc: 0.984
MODEL1 - VAL
Acc: 0.686
MODEL1 - TEST
Acc: 0.725
MODEL2 - TRAIN
Acc: 0.888
MODEL2 - VAL
Acc: 0.710
MODEL2 - TEST
Acc: 0.772
MODEL3 - TRAIN
Acc: 0.875
MODEL3 - VAL
Acc: 0.672
MODEL3 - TEST
Acc: 0.756
Actual
Predicted
cats
dogs
horses
humans


Accuracy Comparison Across Models and Datasets
Model
MODEL1
MODEL2
MODEL3
Accuracy
Dataset
TRAIN
VAL
TEST
F1-Score (Macro) Comparison
F1_Macro
Precision vs Recall by Model
Dataset
Recall
Precision
Evaluation Time by Model
Time (seconds)
Model


F1-Score by Class (Test Set)
Model
MODEL1
MODEL2
MODEL3
0.0
0.2
0.4
0.6
0.8
cats
dogs
horses
humans
Class


Average Per-Class Accuracy (All Models)
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
humans
dogs
cats
horses
Class

Based on the evaluation of the three models (MODEL1, MODEL2, MODEL3) on the Split1 data, the following conclusions can be drawn:

MODEL2 proved to be the best model with a test accuracy of 77.25%, achieving the best balance between performance and generalization thanks to the early stopping training strategy. All models exhibit high overfitting, especially MODEL1 (gap 29.77%), which was intentionally trained with an excessive number of epochs, confirming the effectiveness of regularization techniques in MODEL2 and MODEL3.

Per-class analysis reveals significant differences in classification difficulty - the "humans" class is the easiest (average F1: 87.19%), while "dogs" pose the greatest challenge (average F1: 68.59%), indicating the need to increase data augmentation and balance the training set for this class.

Training strategy has a key impact on results - models with early stopping (MODEL2, MODEL3) generalize better than MODEL1 trained for 100 epochs, despite lower training accuracy. It is recommended to implement ensemble methods and cross-validation to achieve more stable results, as well as to focus particularly on improving the classification of the "dogs" class through targeted data augmentation.

The improve idea

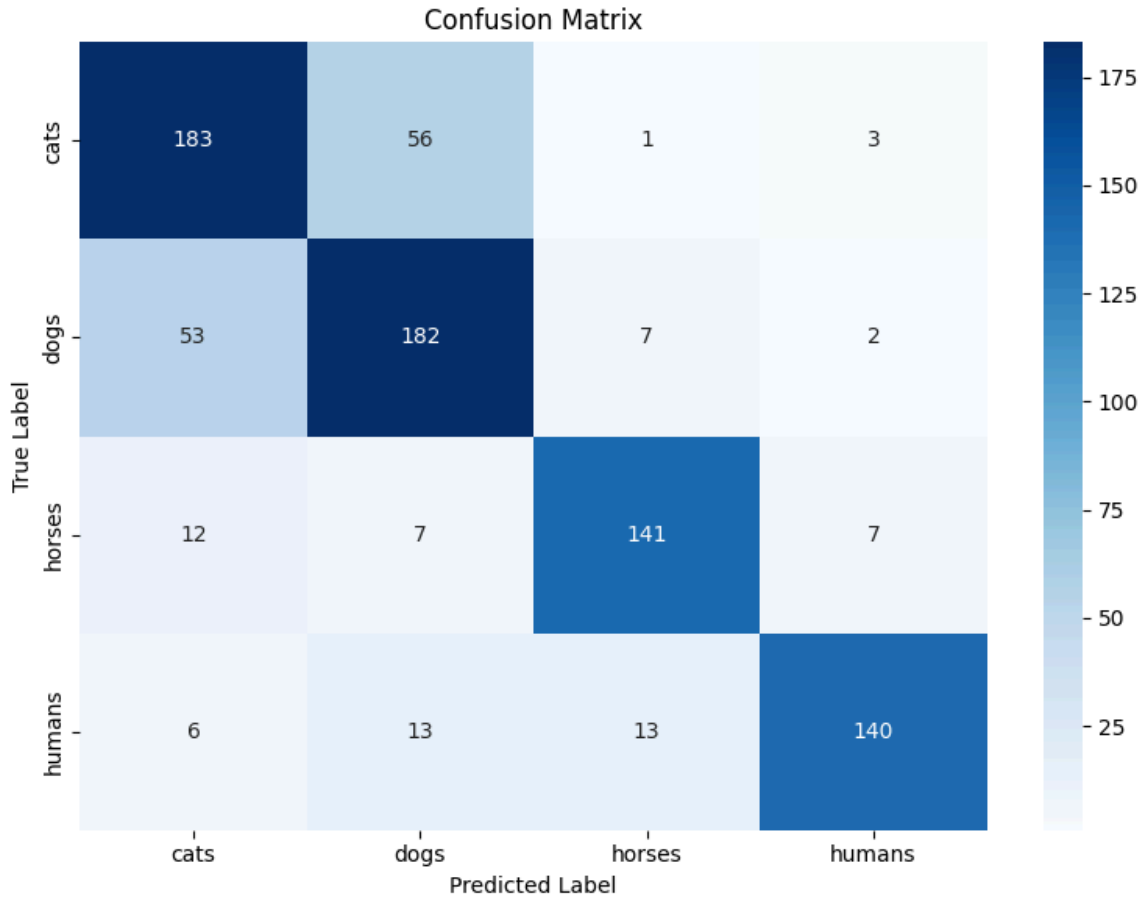
To test how we can improve our model we decided to changes some configuration setting of our models. We decided to see what will happen when we change the amount of augmented images in Split2 dataset. First test were made with 9 augmentation for each image.

We trained two more models on the same dataset as Split2 with the difference that one of them had the augmentation variable set to 4 and the other to 16. As was said in the previous report the ideal amount of Epochs for Split2 was 15 so that was the length of training.

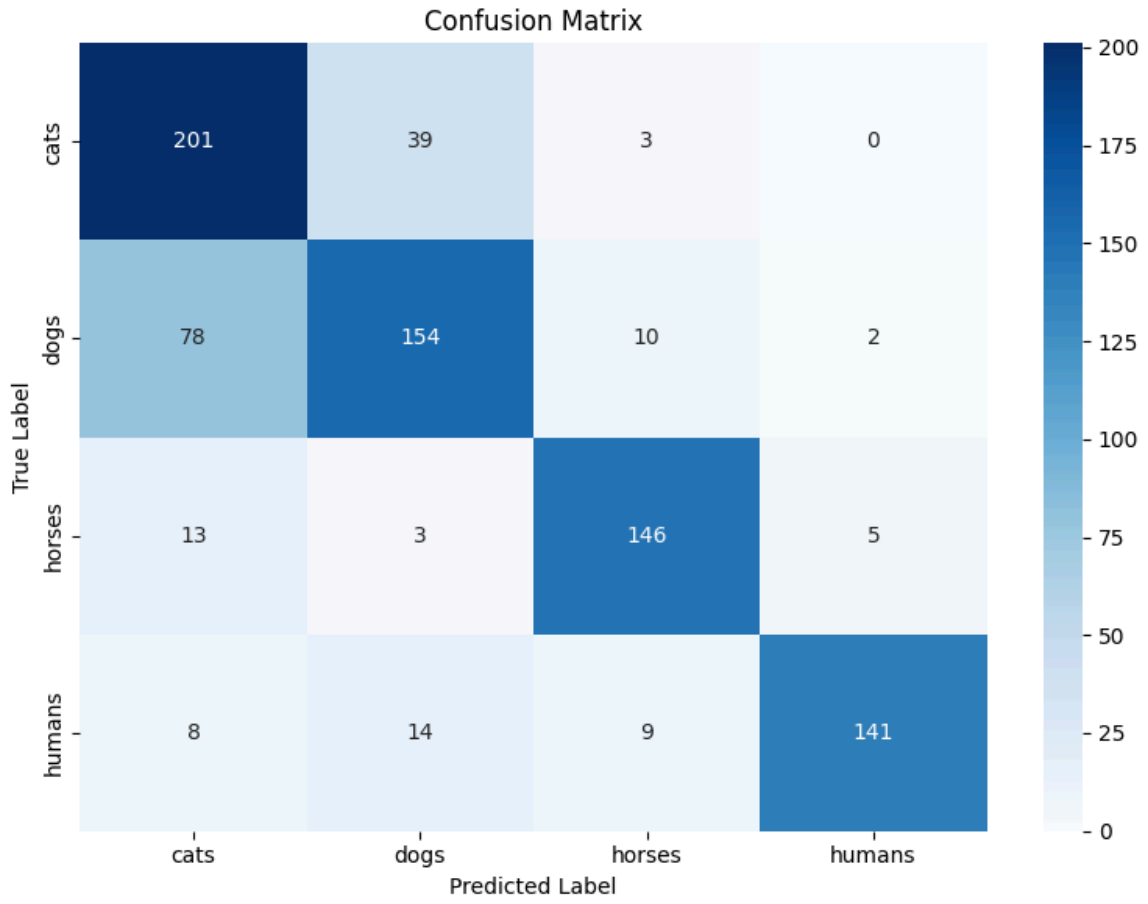
Results of tweaking

Both models did pretty well but there are some nuance that we can discuss.

The graph below shows confusion matrix for the model trained on split2 with only 4 augmentations:



The graph below shows results of training model on Split2 with 16 augmentations:

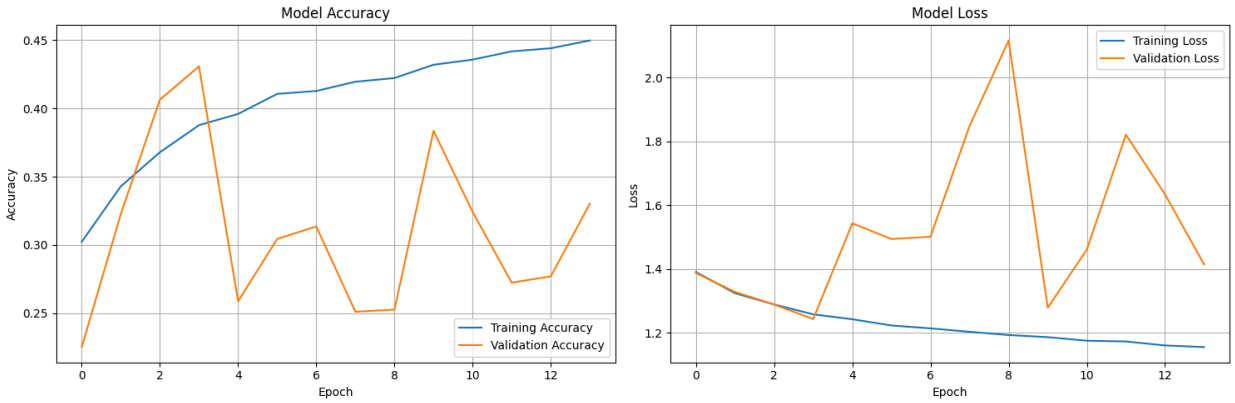


As we can see from this test the model with more augmented images was highly overfitted to classify cats over dogs. The amount of true positives for cats increased but so did the false positives. The amount of dogs mislabeled as cats increased by 10% from the model with 9 augmentations and by 47% from 4 augmentations. This shows that feeding the model with the same images even in amount slightly higher than 4 can very easily mislead the model.

Fine tunning existing model

For tests we also tried to fine tune an already existing model. The EfficientNetB0 which is a neuron network designed for image classification. It has approximately 247 layers. For the purpose of fine tuning we unfroze 30 layers and trained the model on Split2 for 30 epochs.

The results are as follow:

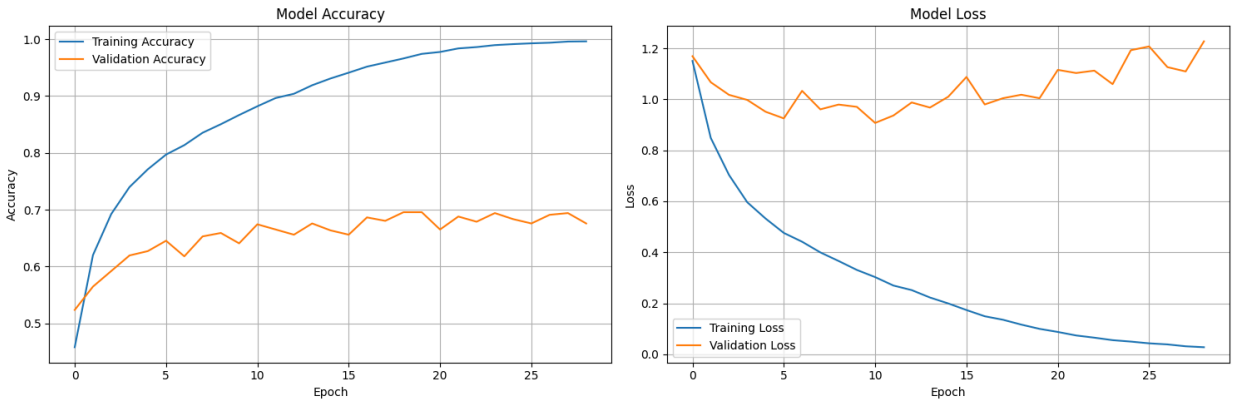


The model stopped training after 14 epochs because early stopping was turned on. The result didn't improve for 10 epochs since epoch 4 (3 on graph)

The model validation accuracy peaked at 0.4307. It's a very poor result. We were not satisfied with the accuracy which we got after a few hours of training so we decided to stick with our own model.

Trying different activation function

On all previous tested models the last layers activation function was "relu". This time we tried to change it to "sigomid" function. Trained our model on Split2 and got this results:



As we can see the validation loss is much lower and doesn't raise much over 1.2 compared to validation loss that we got on Split2 with "relu" that peaked at 2.0.

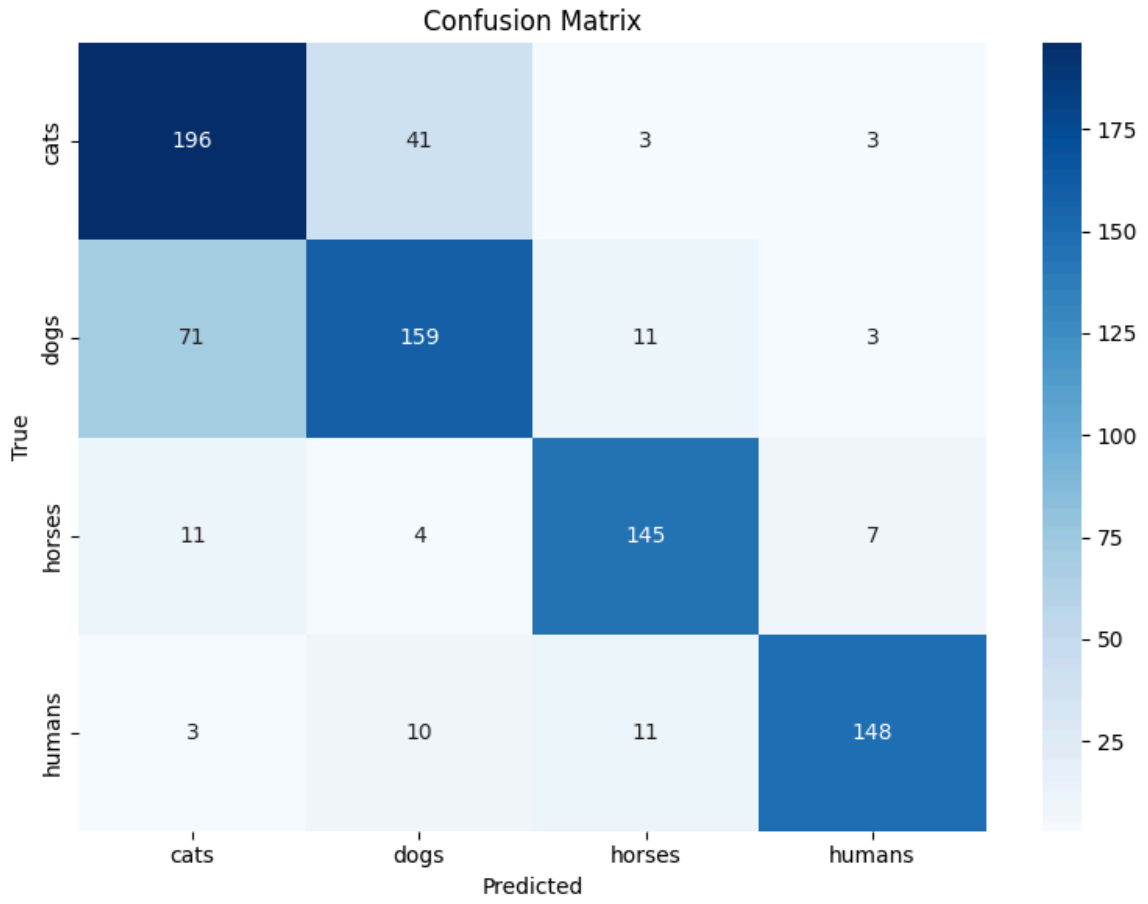
Accuracy peaked on epoch 19 with score of 0.7123. This result is identical to the result we achieved on "relu" on epoch 15.

19th epoch model (best in this training):

- training accuracy: around 0.94
- training loss: around 0.2
- validation accuracy: 0.7123
- validation loss: around 1.14

- 29th epoch model (final epoch):
  - training accuracy: 0.99
  - training loss: 0.02
  - validation accuracy: 0.6757
  - validation loss: 1.2270

The results of confusion matrix are identical to the results we got from using "relu" function. This was predictable because the dataset was the same.



Proposition for further improvement

More data is always better. One of the improvement to our model could be adding more data to dataset. Get more real life images that could actually be tricky for the model to classify and see if after more training it would be capable of classifying these harder images with similar accuracy as now.

As mentioned in previous report. Our training structure could really benefit from early stopping. Right ow when we set the number of epochs, the model will train until it reaches the finish. But as shown on graphs, most of the time it is not necessary. If we implemented early stop we could benefit with saved training time which we could spend on other tweaks and different trainings.

SYSTEMS WITH MACHINE LEARNING

CLASSIFICATION OF ANIMALS
REPORT 3

Adam Cherek, index 207250
Mateusz Nowak, index 189420
Paweł Mańczak, index 188756

We hereby declare that no artificial intelligence (large language models, ChatGPT, etc.) was used to create this report, its text, drawing conclusions, analyzing results and other aspects.

……Adam Cherek… …….Mateusz Nowak…… ……Paweł Mańczak……..

Problem type

Our project is designed as mulit-class image classification problem. Our goal is to assign each input image to our class (cat, dog, horse, human). The model should return the probability for each class and choose the highest one as prediction.

This is a classic example of classification (not regression, detection, or generation).

Training methods

We have pairs of input-output (image and corresponding label - class) so it is **supervised learning.** Unsupervised learning is not suitable because it doesn't use labeled data and it is

used more often for clustering, finding patterns - not for assigning known classes to new given data.

Learning methods

We can divide possible machine learning methods that can be used for image classification into to parts:

Traditional methods: Support Vector Machines (SVM), Random Forests, K-Nearest Neighbors (KNN)

Deep learning methods: Convolutional Neural Networks (CNN), Visual Transformers

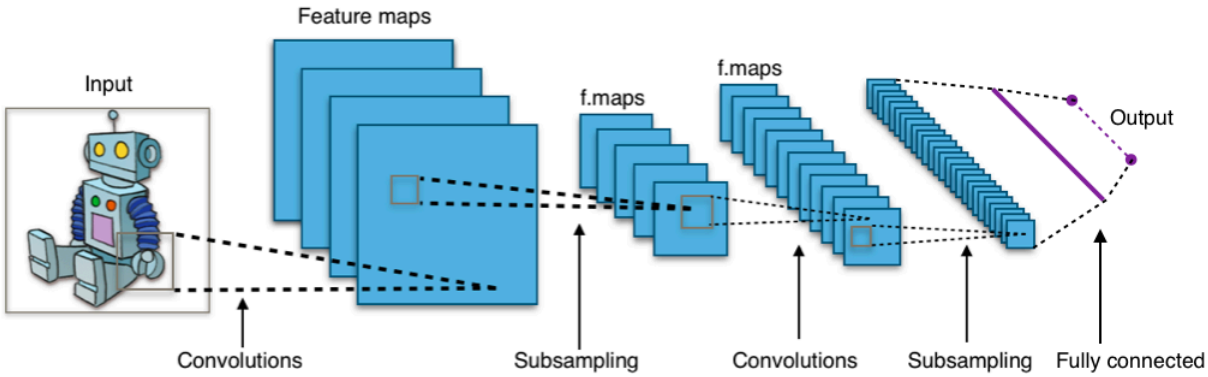
But if we use traditional methods firstly we need to give them some descriptors (they are not designed for catching spatial correlation and for image classification itself) such as Histogram Oriented Gradients (HOG), Scale-invariant feature transform (SIFT).

A pedestrian image (left) and its gradient (right)

https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients

Convolutional Neural Networks (CNNs) are a type of artificial neural network specifically designed for processing data with a grid-like structure, such as images. They use convolutional layers, which can automatically extract important features for us. They move small filters (kernels) across the image which help them to find local patterns, for example edges, shapes.

This means that we no longer need such big image analysis done manually.



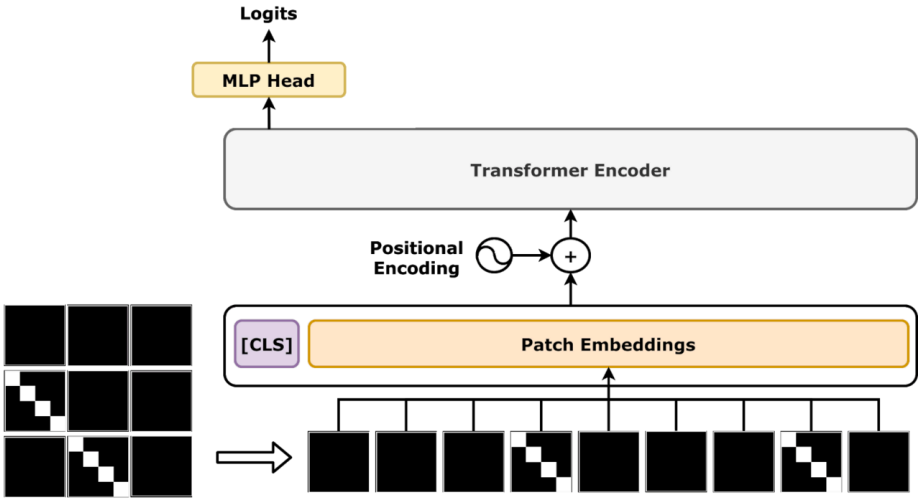
Typical CNN architecture

https://en.wikipedia.org/wiki/Convolutional_neural_network

Visual Transformers (ViT) are a modern artificial intelligence architecture designed for image processing inspired by the Transformer architecture announced by google in “Attention is all you need” article. It is an alternative to traditional Convolutional Neural Networks

ViT works by dividing the input image into a grid of small patches which are treated as sequence of tokens, similar to words in natural language processing (NLP). Every patch is converted into a vector which is enriched with information about position which helps to preserve spatial structure of the image.

The key component of ViT is the mechanism of self-attention, which allows the model to capture relationships between parts of the image. Unlike CNNs, which focus mainly on local features, because of this mechanism Vit can capture both local and global dependencies



The architecture of vision transformer.

https://en.wikipedia.org/wiki/Vision_transformer

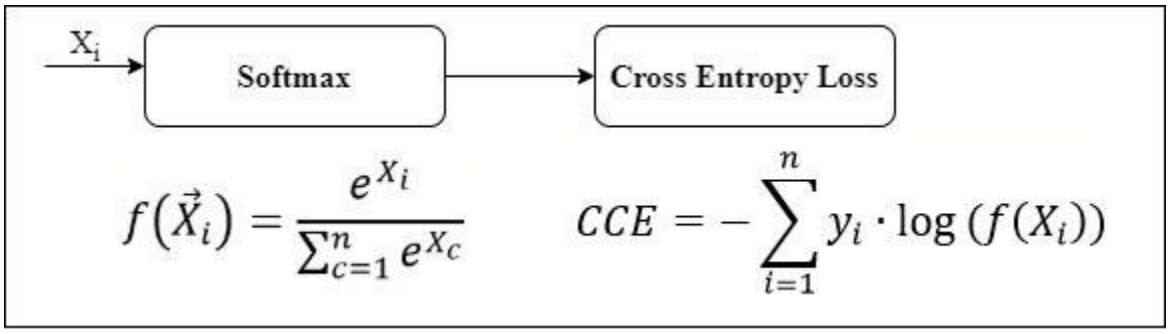
Vision transformers often require larger datasets for optimal performance while CNNs perform well with relatively smaller datasets. Another problem with transformer architecture is that because of self-attention mechanism it requires a lot of computational power, while CNNs with their localized operations are much faster.

That's why we decided to choose conventional neural networks.

Possible loss / fitness functions

For multi class classification tasks the standard loss function is **categorical cross-entropy.**

It measures the difference between the predicted probability distribution (which is output of softmax) and the true label (one-hot encoded vector)

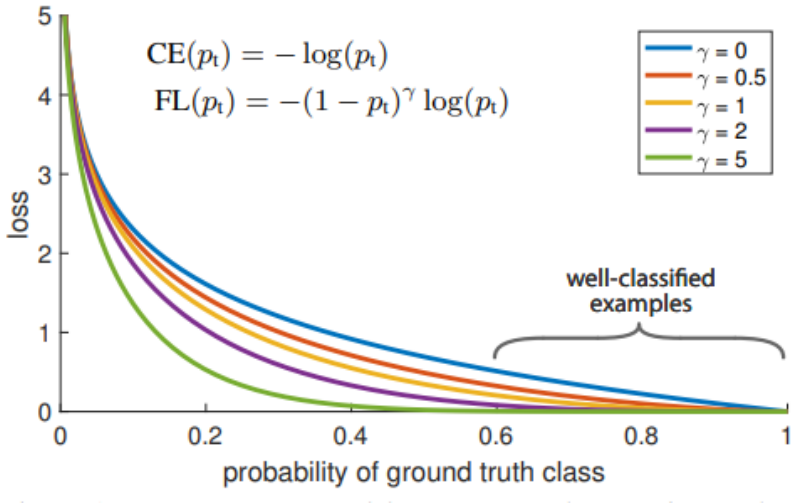


https://www.researchgate.net/publication/368742217_A_customized_1D-CNN_approach_for_sensor-based_human_activity_recognition

Another possible use loss function is Balanced Cross-Entropy Loss which modifies standard cross entropy function by adding weights factors [0.1] to each class which helps dealing with unbalanced datasets. The weight parameter replaces the actual label termin in categorical cross entropy.

Fucal Loss takes this concept further by adding a modulating factor that reduces impact of “easy” example (with high confidence model result) and increases the impact of examples that model misclassified.

The key element is the gamma parameters which should be set by using cross-validation. When the gamma parameter equals zero the Fucal Loss is the same as categorical cross-entropy.



https://towardsdatascience.com/wp-content/uploads/2022/04/1zkWsCJPPtCKzkG1VdZ4atA.png

In practice it is possible to make a variant that takes advantage of both weighting and gamma focusing factor.

Training in general

We performed a training process using TensorFlow and Keras, and used three datasets (Split1, Split2 and Split3) gathered and preprocessed in previous parts of this project. We have created and tested several CNN model architectures in pursuit of higher accuracy and lower overfitting.

Final architecture (the only one which achieved validation accuracy over 70% in Split2) contains:

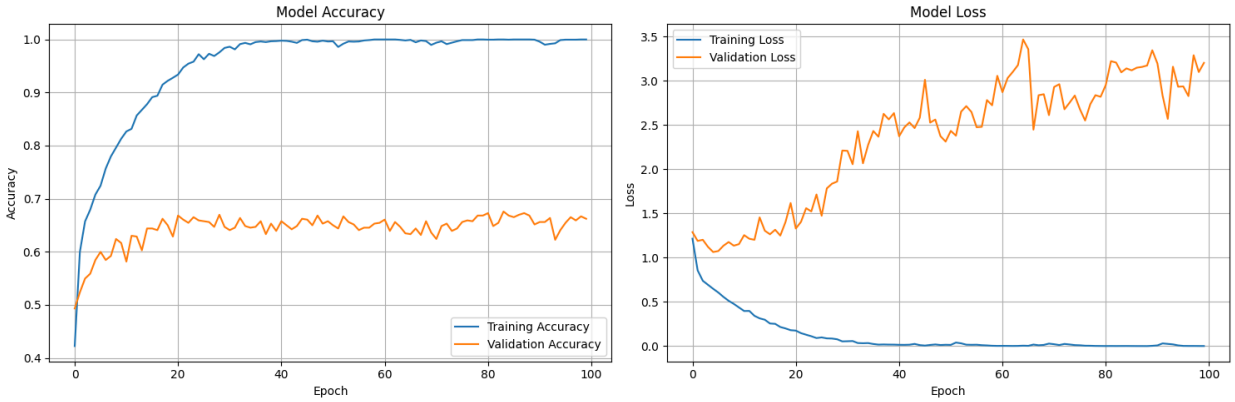
- Input layer: accepts rgb images in 128x128 resolution.
- Extraction blocks: 4 blocks of Conv2D (using 3x3 kernels, ‘same’ padding and ‘relu’ activation) and MaxPooling2D (reducing spatial resolution by half), progressively increasing the number of filters (64, 128, 256, 512).
- Flattening: 3D feature maps are flattened into 1D vector to be fed into dense layers.
- Classification head: Dense(512) with ‘relu’ activation learns high-level combinations of extracted features, then dropout randomly deactivating 50% of the neurons during training to reduce overfitting, and finally Dense with number of classes to classify (4 - cats, dogs, horses, humans), which outputs probability distribution over the target classes.

For compilation we have used Adam optimizer with leating rate of 1e-4, categorical crossentropy loss function and metrics of accuracy during training and validation.

```
def CreateModel(input_shape=(128, 128, 3), num_classes=4):

    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', padding='same', input_shape = input_shape),
        MaxPooling2D(2, 2),

        Conv2D(128, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(2, 2),

        Conv2D(256, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(2, 2),

        Conv2D(512, (3, 3), activation='relu', padding='same'),
        MaxPooling2D(2, 2),

        Flatten(),
        Dense(512, activation='relu'),
        Dropout(0.5),
        Dense(num_classes, activation='softmax')
    ])

    optimizer = tf.keras.optimizers.Adam(learning_rate=1e-4)
    model.compile(
        optimizer=optimizer,
        loss='categorical_crossentropy',
        metrics=['accuracy']
    )

    return model
```

Training on Split1

We performed the training on the train subset of the Split1 dataset (original set without any augmentation) and simple testing on the validation subset. To cause model overfitting we disabled early stopping of training and went through 100 epochs in time of 5346 seconds (1h 29min).



Normally our early stopping would stop training in epoch 39 (38 on plot as it is iterated from zero) due to no improvement in validation accuracy in the last 10 epochs and the best model would be the one from the 29th epoch (28th on plot). However, since we have performed 100 epochs of training, surprisingly the final best model (in terms of validation accuracy) of training on Split1 was the one from the 84th epoch (83 on plot).

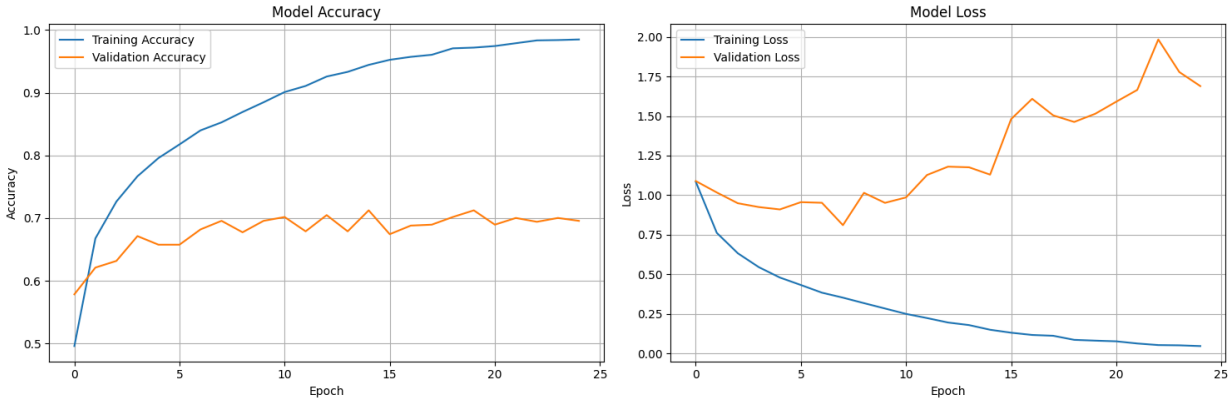
- 29th epoch model (best with early stopping):
    - training accuracy: around 0.99
    - training loss: around 0.1
    - validation accuracy: 0.6697
    - validation loss: around 2.1
- 84th epoch model (best in this training):
    - training accuracy: 1.0
    - training loss: 0.0
    - validation accuracy: 0.6758
    - validation loss: around 3.2
- 100th epoch model (final epoch):

- training accuracy: 1.0
- training loss: 0.0
- validation accuracy: 0.6757
- validation loss: 3.204

Shape of the model loss plot suggests that model overfitting may be more than significant but we will check it deeper during the 4th part of the project.

Training on Split2

We performed the training on the train subset of the Split2 dataset (containing augmentation) and simple testing on the validation subset. This time our early stopping method was turned on, so the training ended after 25 (11723 seconds = 3h 15min) of 50 epochs.



This time the plot looks much better but at the validation loss it can be seen that despite the attempt to limit overfitting, it probably occurred again. Still, effects of this training are much better than those of Split1 training. Model2 (15th epoch, 14th on plot) has higher validation accuracy and lower validation loss.

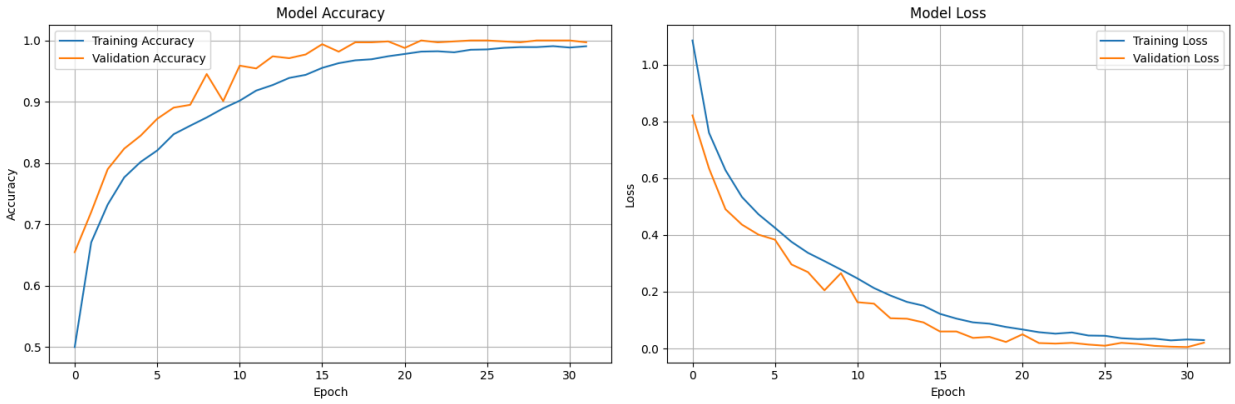
- 15th epoch model (best in this training):
  - training accuracy: around 0.94
  - training loss: around 0.2
  - validation accuracy: 0.7123

    - validation loss: around 1.14
- 25th epoch model (final epoch):
    - training accuracy: 0.98
    - training loss: 0.04
    - validation accuracy: 0.6955
    - validation loss: 1.6894

In direct comparison of the best models of Split1 training and Split2 training we can observe that validation accuracy is 0.365 is higher in favor of Model2 and validation loss is around 2,06 (nearly 3 times) lower again in favor of Model2. On this basis we can conclude that data augmentation (additionally reinforced by the final architecture of our model) had a significant positive impact on our training. Further tests will be carried out in the 4th stage of this project.

Training on Split3

Finally we performed the training on the train subset of the Split3 dataset (set similar to Split2 but with the validation subset replaced with part of the training subset) and simple testing on the validation subset. Again, an early stopping method was turned on so the training ended after 32 (14964 seconds = 4h 9min) of 50 epochs.



In line with our predictions, changing the validation subset with part of the training subset led to accuracy and loss plots being similar in both training and validation. At first glance it

might seem that convergence has been achieved and the model has reached its optimal state, but due to validating on part of training data the real performance of models trained this way is a mystery. Anyway, we had to choose the best model from this training and it turned out to be a model from the 22nd epoch (21 on plot).

- 22th epoch model (best in this training):
    - training accuracy: around 0.99
    - training loss: around 0.08
    - validation accuracy: 1.0
    - validation loss: around 0.01
- 32th epoch model (final epoch):
    - training accuracy: 0.99
    - training loss: 0.03
    - validation accuracy: 0.9969
    - validation loss: 0.0204

In direct comparison of Model2 (best model of training on Split2) and Model3 (best model of training on Split3) someone might say that Model3 is a miracle - always right (validation accuracy 1.0) with high confidence of right answers (validation lost around 0.01), while Model2 is not the brightest bulb - only 71% of correct answers and loss indicating the probability of high confidence in wrong answers and low confidence in right ones, but the truth will be revealed in further testing (part 4 of the project) which the quality of training with such an absurd condition as validating on same data as used for training, may show that Model2 is far superior to Model3.

Final conclusions

Just as we thought most promising is Model2 (training on Split2) but everything will be decided only after final tests in stage 4. Despite all efforts our models seems to overfit more than we expected and accuracy is not as high as we would like it (we wanted at least 75%) , but taking into account that random classification of 4 classes would have 25%

accuracy, our 71% with Model2 is still not a bad result for the resources used and the relative simplicity of our models.

SYSTEMS WITH MACHINE LEARNING

CLASSIFICATION OF ANIMALS
REPORT 2

Adam Cherek, index 207250
Mateusz Nowak, index 189420
Paweł Mańczak, index 188756

We hereby declare that no artificial intelligence (large language models, ChatGPT, etc.) was used to create this report, its text, drawing conclusions, analyzing results and other aspects.

……Adam Cherek… …….Mateusz Nowak…… ……Paweł Mańczak……..

Data description

For training and testing, we will need:

- Input - raw images. Preferably in jpg format, color (RGB), resolution is not that important, because we will downsize it anyway, but for performance reasons it should be not bigger than 10000x10000. We also won't take images smaller than 64x64. It should be square, but it is not mandatory. The images shall contain only natural objects, drawings etc..
- Output - correct classification labels assigned to input raw images, easy to read.

We plan to collect data from Kaggle datasets. If it will be hard to achieve decent results we will bring together more datasets, but the aim is to use as small as possible.

Dataset content

We used following data sources:

- *https://www.kaggle.com/datasets/osamaadelelsayed/catsdogshorses-and-humans/data*
- *https://www.kaggle.com/datasets/tongpython/cat-and-dog/data (only test set)*
- *https://www.kaggle.com/datasets/sanikamal/horses-or-humans-dataset*
- *for further testing we will manually crawl from Google Images*

Data statistics:

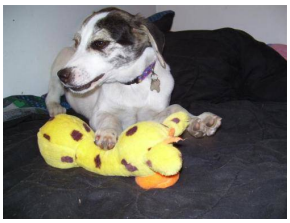
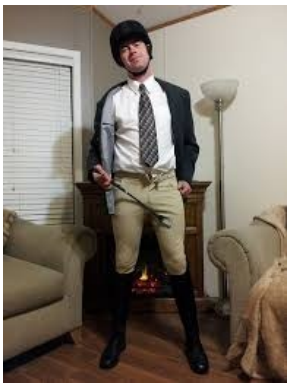
Data source	**Classes**	**Total count**	**Format**	**Data distribution**
https://www.kaggle.com/datasets/osamaadelelsayed/catsdogshorses-and-humans/data	cat, dog, horse human	808	various resolutions, jpg, RGB	uniform class distribution
https://www.kaggle.com/datasets/tongpython/cat-and-dog/data *(only test set)*	cat, dog	2025	various resolutions, jpg, RGB	uniform class distribution

https://www.kaggle.com/datasets/sanikamal/horses-or-humans-dataset	horse, human	1283	various resolutions, png, RGB	uniform class distribution

Whole dataset contains **4116 of single, RGB images**, almost evenly distributed into 4 categories: **dog, cat, horse, human**. As a result, there are 800-1200 images per category. This picture below represents a sample (randomly chosen) from our data with **labeling done by the creators** of the sourced dataset. **Each image belongs only to one category** and the labeling is in **boolean** form (no percentage of how much a certain image looks like for example dog, just to which category each image belongs to). But there is a possibility that pictures for example shows a cat which is in the hand of human.

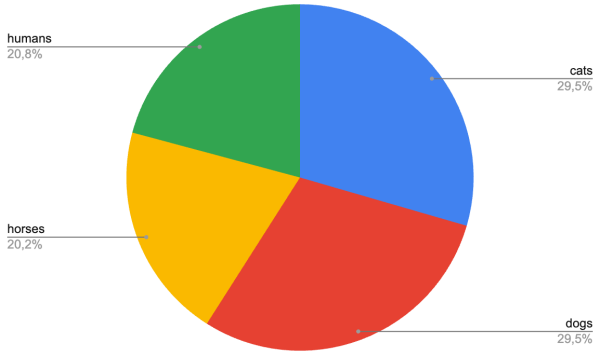
Cat	Dog	Human	Horse
			

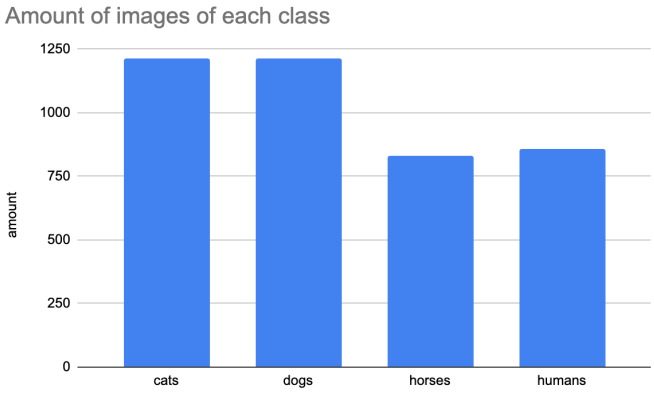
			
			
			

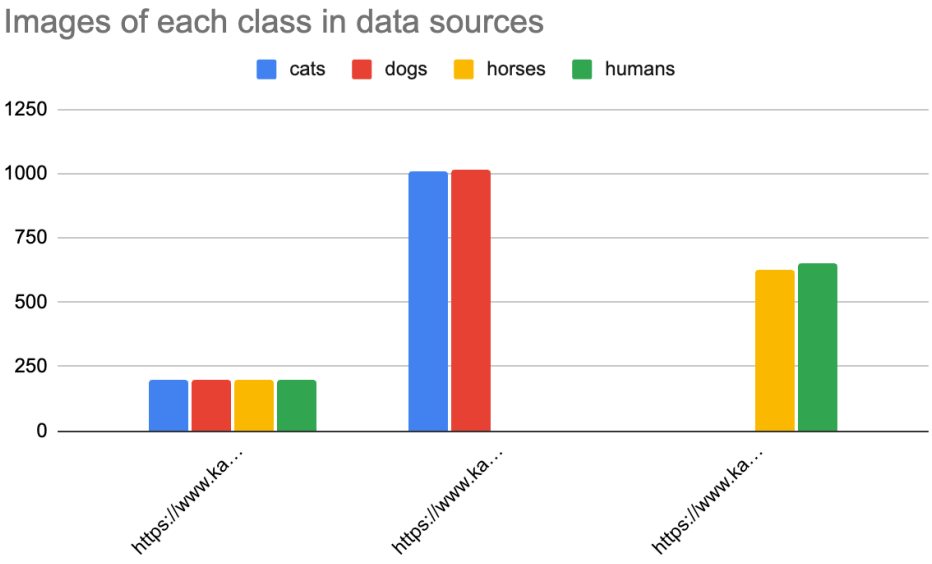







The charts below illustrate the amount of data and proportion of each category in the dataset. As we can see, the dataset is not entirely balanced, but the difference of number of images for each class is not so big. This even distribution ensures that our solution will not struggle with correctly classifying "rare" categories and will not be biased toward certain categories due to a larger number of images in those classes.


humans
20,8%
cats
29,5%
horses
20,2%
dogs
29,5%


Amount of images of each class
amount
1250
1000
750
500
250
0
cats
dogs
horses
humans


Images of each class in data sources
cats
dogs
horses
humans
1250
1000
750
500
250
0
https://www.ka...
https://www.ka...
https://www.ka...

Data relations

In the “horse” group we can see some repetitions, for example there are a lot of photos of the same horse but from slightly different perspective. There are photos of heads, whole bodies (sometimes in movement) of various kinds of horses . So finding patterns in this group of horses photos should be relatively easy for our models. We assume that besides some similar photos this will not affect training or testing of the model, so **no action is needed**.

Same horse from different perspectives example:

Similar with humans:

On the other hand, distinguishing between the "dog" and "cat" classes can be quite tricky. These two categories share a lot of visual similarities. Both often appear in images with similar backgrounds (e.g. white wall), and both have similar features like fur and body posture. A helpful aspect is that dog images often contain unique clues such as longer snouts, collars, or leashes, which might assist the model during training.

Difference between humans and dogs is usually pretty big, but in our data we have images that may case some problems such as photo of dog in policeman uniform:

Images classified as **“dog” and “cat” are in most cases unique**. Characteristics should be easily recognized. No extra edition in most cases needed.

We did not find any exact duplicate images in the dataset, we assume that there is no repeated data, but it is impossible to go through all the data so we check only a part. However, we did observe a significant number of visually similar images in some cases, showing the same object or subject captured from slightly different angles or in different lighting conditions. The presence of such similar images varies depending on the data source. The horse-human dataset tends to have a bigger amount of near-duplicates, while other images don’t have such problems

There is a risk related with background of photos, but it could be both a risk and a benefit: the model might learn to rely on background instead of the actual object, but in real-world use, context can be helpful (e.g., humans are more likely indoors than cats on trees).

In our case, similar images appear across different data sources with no clear distinction in terms of resolution, color, or class distribution. Because of this, we decided not to separate the data by source and instead created a single, unified test set. This simplifies evaluation and avoids splitting already similar data into artificial groups.

Errors and noise

> ***Analyze noise and errors*** *in the data (types of noise, types and number of errors, wrong labels, etc.) Present some examples.*

In our dataset we can stand upon many examples of **blurred or low resolution images.** This can make the classification much harder or even mislabel the class. For example an image of small dog can be misinterpreted as a cat.

There could also occur **background clutter effect**. When the main target on image is placed near another analyzed class. Example below shows an image labeled as a cat but the main target is held in human hands which can confuse the algorithm. **This kind of images are in minority** so it's effects can be neglected.

Data sources that we plan to use are from trusted source so we can exclude the risk of wrongly labeled images.

For some images it will be necessary to apply data augmentation such as rotation, cropping and brightness adjustment to improve robustness. More about it in the next chapters.

Data difficulty

***Check if some data is "more difficult" or "more important" than the other**, decide if you need to treat it separately (e.g. weighting during training or separate test sets).*

Most of the dataset consists of high quality, clear images. Almost all of the images belong unambiguously to one of the defined classes (with some exceptions like animals held in human hands).

However, there are some (less than 1%) ambiguous images, possibly containing objects belonging to more than one class. Example of such ambiguity is described in the previous section.

Some of the photos are close-ups of the considered objects (especially dogs, presented in the image below). Such photos are common in the dataset (at least 1%) - we don't anticipate that this will be a problem for either training or testing, thus we do not treat them differently than the rest of the dataset.

Some images may carry **more useful information for learning than others.** For instance, a clear image of a dog in good lighting is easier for the model to learn from. Also an image showing a rare pose or a mixed group of animals could provide more nuanced examples that help the model better generalize. It would be suggested to **prioritize** or **give more weight** to those more informative or rare samples during training

Some images are **naturally more difficult to classify** correctly **due to lighting conditions** (shadows or overexposure may obscure features), **poses** of the animals and humans, **occlusion** (part of the target may be blocked or cropped out) and **similarities** such as a

fluffy cat may appear to neural network as a small dog, **increasing risk of wrongful classification**.

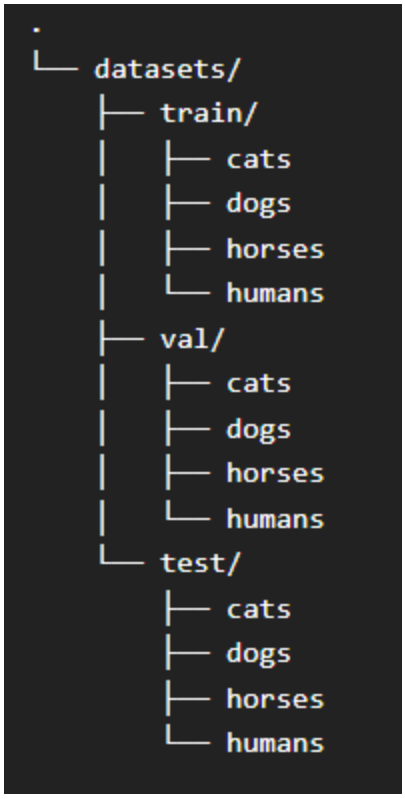
By identifying which images are harder or more important we can apply **special preprocessing** (explained more in next chapters) or apply weighted loss functions (giving more penalty when model gets those wrong) which was mentioned in previous chapter.

There also could be a difficulty with the “other” class. Some images may have features of analyzed targets but are completely different things. For example a fluffy tiger which should go under the class “other” could have very high score in “cat” category which may confuse the model despite the possibility of “other” class.

Data representation

> ***Choose a method of data representation*** *(raw image data, features, spectrogram, dictionary, word2vec, etc.) Convert the data*

We plan to structurize the data into 3 categories: train, val and test. Each category has its own folders for cats, dogs, horses and humans which contain images. The scheme below visualizes the data placement:



In directory we change the names of files to fit the format as follows for original photos:

<class name>_<index>.jpg

Our photo import function is parametrized so we can easily change the format of data naming and also the image extension (jpg).

For augmented photos we will use this format:

<class name>_<index>_<augmentindex>.jpg

Each image is represented as a 3D array of pixel intensity values. We decided to use grayScale format. A 128x128 image with 1 color channels will be represented as a tensor of shape [128, 128, 1]. All of the images will be normalized so they have the same size and color mode. More on that in next chapter. As for now we don't have plans to use any feature extractions.

Data normalization

> ***Propose and perform data normalization*** *and other preprocessing (if applicable to your subject).*

Our propositions for data *normalization* include:

- **scaling each pixel value to a new value from range <0; 1>.**
- **changing color mode to "greyscale"**

Those changes should reduce complexity of data and allow faster training of our model.

We have also decided to *standardize* our data by:

- **setting all images to the same size.**
- **setting all images to the same extension.**

We believe it could stabilize and potentially accelerate the training process. Unscaled data may lead to large weight values in solutions based on for example neural networks which may cause instability of the model.

We have also prepared some functions performing *normalization and standardization* of our dataset. The one responsible for normalization is shown in the picture below, this method performs normalization with use of keras_preprocessiing.image library, while loading a dataset from a directory.

```
22
23  def LoadDatasetWithNormalization(source_dir, batch_size = 32):
24      print(f"Loading data from {source_dir}")
25      datagen = ImageDataGenerator(rescale=1./255)
26      dataset = datagen.flow_from_directory(
27          source_dir,
28          batch_size=batch_size,
29          color_mode='grayscale',
30          class_mode='categorical',
31          shuffle=True
32      )
33      return dataset
34
```

Data augmentation

Propose and perform simple data augmentation *(if applicable to your subject).*

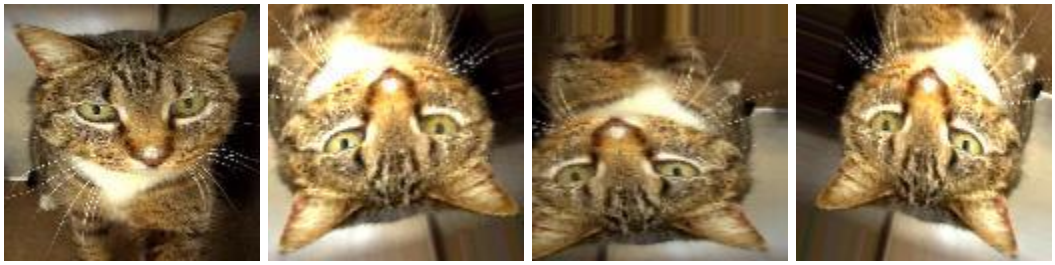
For **data augmentation**, we used the class "ImageDataGenerator" which is available in previously mentioned keras_preprocessing.image Python library. There are several parameters that are accessible in the mentioned class. We choose to:

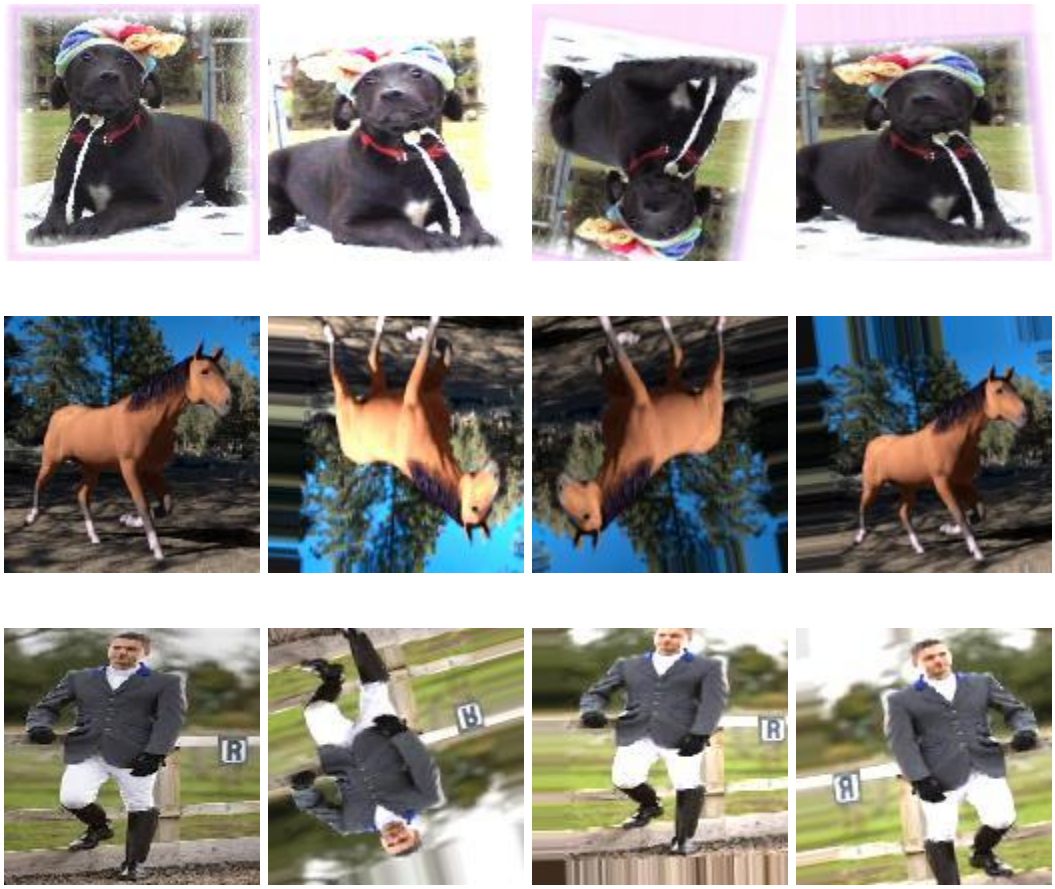
- **shift images both vertically and horizontally**
- **flip images both vertically and horizontally**
- **rotate images**
- **zoom images**
- **change brightness**

All of those operations are performed randomly by ImageDataGenerator, after performing augmentation we save original image and augmented copies to the dataset directory. The quantity of augmented copies depends on variable so later in the project we will be able to observe if augmentation really helps in our case and if so, how many copies is optimal. Part of the code of our augmentation method is presented below.

```
def StandardizeAndAugmentData(source_dir, img_size=(128, 128), augmented_copies = 1, extension = ".jpg"):
    #augmentation for training
    train_datagen = ImageDataGenerator(
        rotation_range=20,
        zoom_range=0.15,
        width_shift_range=[-10, 10],
        height_shift_range=[-10, 10],
        horizontal_flip=True,
        vertical_flip=True,
        brightness_range=[0.5, 2.0]
    )

    #augmentation for test/val (no augmentation)
    simple_datagen = ImageDataGenerator()
```

Sample augmented photos are presented below (original image is first from the left)





Data splits

> ***Split the data into TRAIN/VAL/TEST sets*** *(if applicable to your subject). Calculate and plot some statistics about them. Create 3 different data splits:*

The whole dataset was split into training, validation and testing subsets, as required in the project instructions. Three different such splits were created. To split originally downloaded datasets, we used our function presented below. It's quite long because not only does it split the original dataset but also copy splitted subsets to created directories for split1, split2 and split3 as we like to have our datasets physically on disk.

```python
def CopySplitDataFromDir(source_dir, output_dir, train_ratio=0.64, val_ratio=0.16, test_ratio=0.20):

    #remove and recreate split dir
    if os.path.exists(output_dir):
        shutil.rmtree(output_dir)
        print(f"Removed existing '{output_dir}/'")
    os.makedirs(output_dir, exist_ok=True)

    classes = [cls for cls in os.listdir(source_dir) if os.path.isdir(os.path.join(source_dir, cls))]

    #coping and spliting
    for cls in classes:
        cls_path = os.path.join(source_dir, cls)
        images = os.listdir(cls_path)
        images = [img for img in images if os.path.isfile(os.path.join(cls_path, img))]

        n_total = len(images)
        n_train = int(n_total * train_ratio)
        n_val = int(n_total * val_ratio)

        #split images to sets
        train_imgs = images[:n_train]
        val_imgs = images[n_train:n_train + n_val]
        test_imgs = images[n_train + n_val:]

        counters = {
            'train': 1,
            'val': 1,
            'test': 1
        }

        #create splits and classes directories and copy splited images
        for split_name, split_imgs in zip(['train', 'val', 'test'], [train_imgs, val_imgs, test_imgs]):
            split_class_dir = os.path.join(output_dir, split_name, cls)
            os.makedirs(split_class_dir, exist_ok=True)

            for img in split_imgs:
                src = os.path.join(cls_path, img)
                ext = os.path.splitext(img)[1].lower()  #keep extension
                new_name = f"{cls}_{counters[split_name]}{ext}"
                counters[split_name] += 1

                dst = os.path.join(split_class_dir, new_name)
                shutil.copyfile(src, dst)

    print(f"{output_dir} complete")
```

We did not change the distribution of data between classes, as already at the data collection stage we tried to make the classes more or less equal.

Split 1

> ***SPLIT 1.*** *Original data (no normalization, no augmentation), unmodified class distribution / proportion*

First data split using original data without much modifications. No data augmentation, normalization, standardization was used as you can see on the previous picture of the splitting function. We decided to put 64% of images into the TRAIN set, 16% into the VAL set and the remaining 20% into the TEST set. As a result we get 2468 images in the TRAIN set, 616 in the VAL set and 776 in the TEST set. Of course, the proportions can be adjusted later to achieve potentially better results.

Split 2

> ***SPLIT 2.*** *Properly normalized and augmented data, proper class distribution among train/val/test sets*

In the second split, data is required to be correctly preprocessed, and class distribution should be similar in both training, validation, and testing sets. As we mentioned before our **dataset originally contains a similar number of images for every class**, only image preprocessing (standardization, augmentation and normalization) has to be done.

Our previously presented function already copied and splitted original dataset so now we just use the standardization and augmentation function (presented before in augmentation section) on split2 directory and after a while its ready to load with our previously presented normalization function utilizing flow_from_directory method of keras_preprocessing.image ImageDataGenerator .

Split 3

> ***SPLIT 3.*** *Like "SPLIT2", but instead of original VAL set use part of TRAIN set, keeping TRAIN and TEST the same as before*

Creating SPLIT 3 set only required manual copying of part of the split2 train set and switching it with validation set inside split3 directory, but we still created a function (presented below) for this need in case we need to do it more often later in the project . Then same as before we can load the dataset with our normalization function.

```python
def CreateSplit3FromSplit2(split3_dir, split2_dir):
    # Remove and recreate split3_dir
    if os.path.exists(split3_dir):
        shutil.rmtree(split3_dir)
        print(f"Removed existing '{split3_dir}/'")
    shutil.copytree(split2_dir, split3_dir)
    print(f"Copied contents from '{split2_dir}' to '{split3_dir}'")

    train_dir = os.path.join(split3_dir, 'train')
    val_dir = os.path.join(split3_dir, 'val')

    # For each class, replace val with part of train (copy only, no deletion from train)
    class_names = os.listdir(val_dir)
    for cls in class_names:
        train_cls_dir = os.path.join(train_dir, cls)
        val_cls_dir = os.path.join(val_dir, cls)

        train_imgs = os.listdir(train_cls_dir)
        val_imgs = os.listdir(val_cls_dir)
        val_imgs_count = len(val_imgs)

        if val_imgs_count == 0:
            print(f"Skipping class '{cls}' due to no val images")
            continue

        if len(train_imgs) < val_imgs_count:
            print(f"Not enough train images to copy for class '{cls}' (train: {len(train_imgs)}, val target: {val_imgs_count})")
            continue

        # Clear val directory
        for file in val_imgs:
            os.remove(os.path.join(val_cls_dir, file))

        # Copy first n_val images from train to val
        for i in range(val_imgs_count):
            src = os.path.join(train_cls_dir, train_imgs[i])
            dst = os.path.join(val_cls_dir, train_imgs[i])
            shutil.copyfile(src, dst)

        print(f"Copied {val_imgs_count} train images to val for class '{cls}'")

    print(f"{split3_dir} ready with copied validation set")
```

SYSTEMS WITH MACHINE LEARNING

At the end of Report 2, please include your Report 1 (!!!)

SYSTEMS WITH MACHINE LEARNING

CLASSIFICATION OF ANIMALS REPORT 1

Adam Cherek, index 207250
Mateusz Nowak, index 189420
Paweł Mańczak, index 188756

Problem definition

The system will recognize cats, dogs, horses and humans on images (photos or video frames). If no cat/dog/horse/human is present in the image, the system will return “none” as an answer.

Number of specified specimens on image may matter, as most of training and testing images will contain only one specimen. If multiple cats, dogs, horses or humans are present in the image, the system should still correctly categorize the image as cat/dog/horse/human image but probably with less accuracy .

If multiple different animal breeds are present in the image, the system's behaviour is not specified.

System input

Raw images of size at least 200 x 200 pixels. Images will be saved using lossy compression as JPEG files. Images will be mostly color: RGB colorspace, 8-bit pet color.

Training and testing data will be mostly from kaggle.com/datasets website. If necessary, the rest will be collected from the internet manually or using a web scraper.

Each image will contain either: cat(s), dog(s), horse(s), human(s) or none of them.

No special image pre-processing stages are planned. Images will be only slightly modified (normalization, augmentation). Raw pixels will be the input for machine learning algorithm.

Examples of input data:




System output

The system will return the class of the image. There will be 5 possible classes (labels):

1. Cat - the image represents cat or cats
2. Dog - the image represents dog or dogs
3. Horse - the image represents horse or horses
4. Human - the image represents human or humans
5. None - the image does not contain any of specified specimens

Ground truth data (correct labels) will be collected during downloading data set from keggle.com/web crawling and also by manual labeling.

There is no planned post-processing of machine learning output. Raw confidence (probability) values for each class will be returned. Therefore, the system will have five numerical outputs, one for each class. Each value will be a real number in range [0,1], where 0 = 0% confidence, and 1 = 100% confidence.

Example outputs for inputs:

	Cat: 0.01 Dog: 0.01 Horse: 0.02 Human: 0.95 None: 0.01
	Cat: 0.03 Dog: 0.03 Horse: 0.91 Human: 0.02 None: 0.01
	Cat: 0.89 Dog: 0.05 Horse: 0.03 Human: 0.01 None: 0.02