Deep Learning – Assignment 1 - Research Summary Report

Submitted by:

Itay Bouganim, 305278384

Ido Rom, 312239858

Shauli Genish, 311242150

Overview

The dataset we worked on during the assignment was Kaggle's dog breed data set which consists of 120 labels of dog breeds. We are provided with a training set and a test set of images of dogs. Each image has a filename that is its unique id. The dataset comprises 120 breeds of dogs. The goal is to create a classifier capable of determining a dog's breed from a photo. The list of breeds is as follows

1. DATASET EXPLORATION & ANALYSIS

1.1 What is the size of the data

We started our research into Kaggle's Dog Breed dataset by examining the amount of data available to us and its distribution between training data and testing data.
The distribution was as follows:

Train dataset size: 10222
Test dataset size (unlabeled): 10357
Total classes: 120

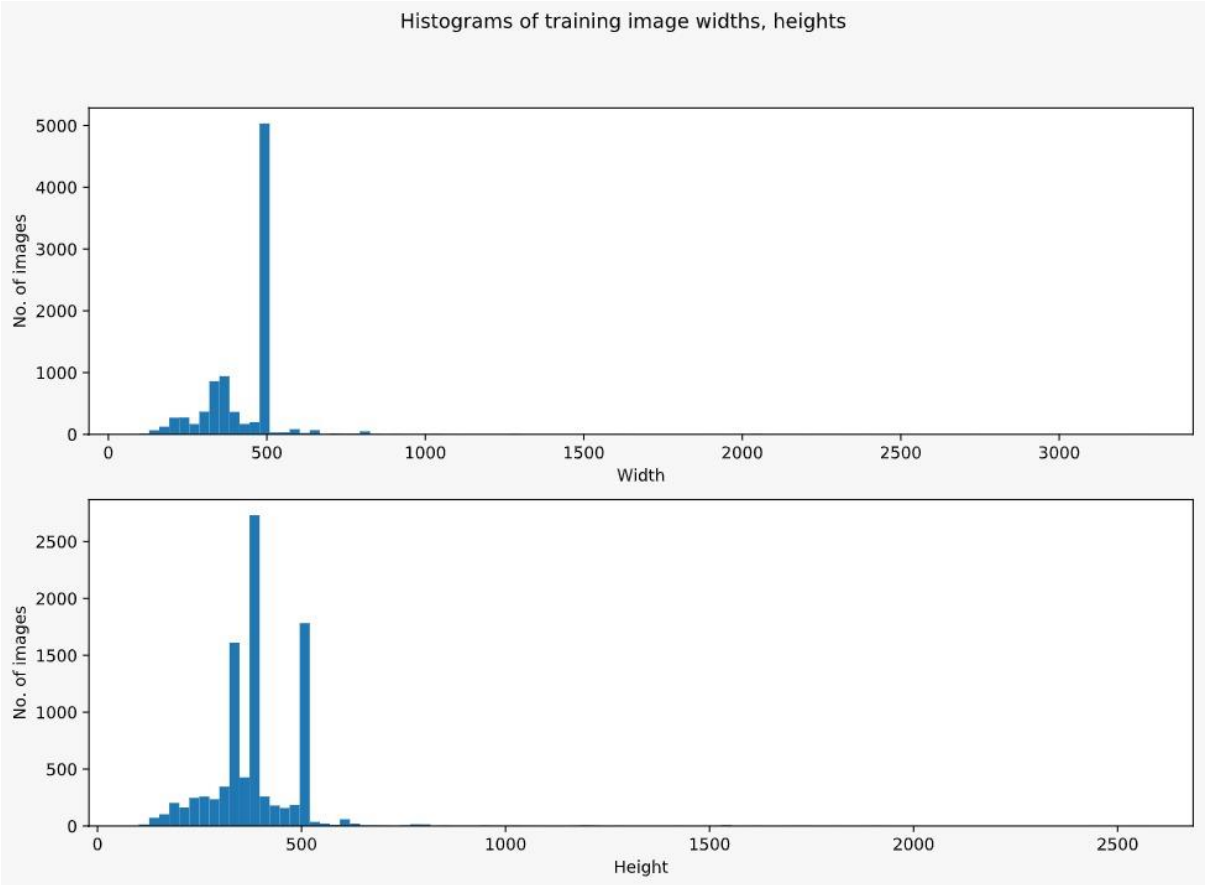
We found out that the dataset does not contain labels for the testing data, A fact that will later make it harder for us to test our models performance in a way that will reflect the entire data. After each iteration of the model we submitted our results as CSV to Kaggle to determine our score on the testing data.

1.2 What data does each sample contain

The data samples varied greatly, in quality, dimensions, viewing angles and representation of the labels. Some of the samples had more than one entity in them while others had a blocked view of the sample or contained a lot of noise that can make the learning process harder.

After analyzing the samples dimensions those were the results:

Training data average (width, height): (443.22055327425977, 387.07974929760104)
Training data (minimum width, maximum width): (97, 3264)
Training data (minimum height, maximum height): (102, 2562)
Training data (Standart deviation of widths, Standart deviation of heights): (152.3010044037075, 129.97994606217097)

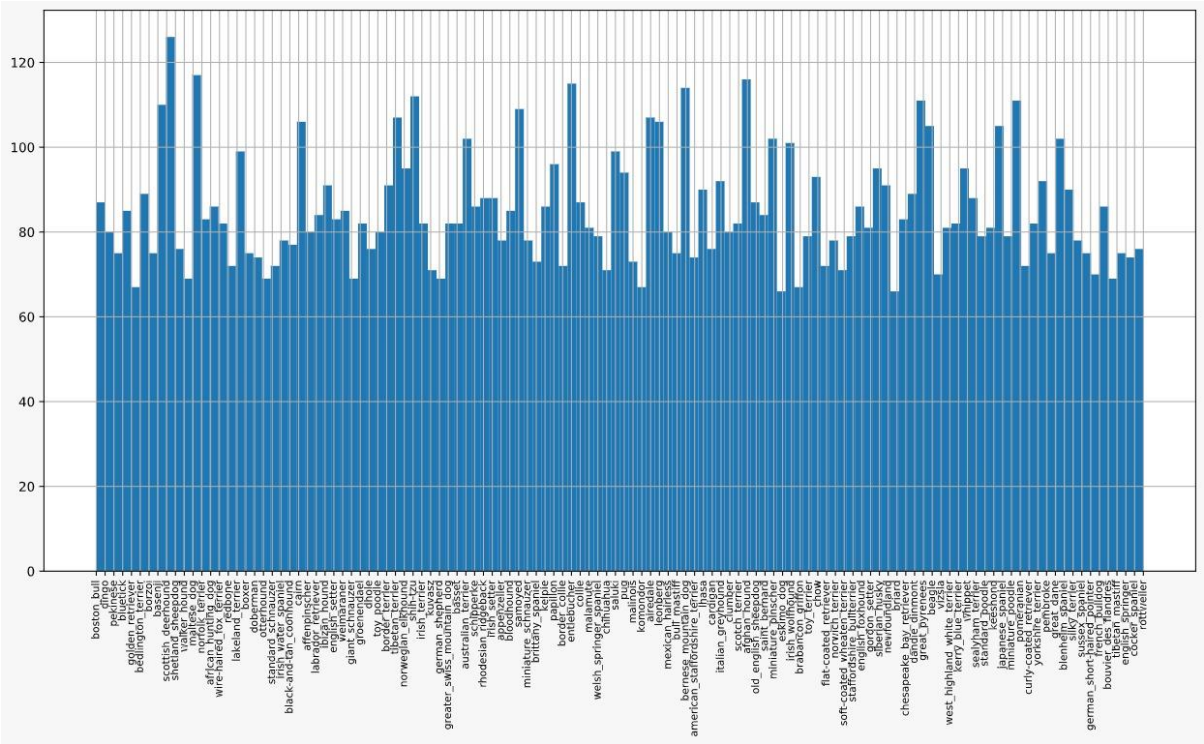


Since the samples are in different in dimensions, it is necessary to resize them to a uniform size. We will do it using generators at inference time and will leave it be at that stage. We chose to convert all images to 128X128X3.

We can use augmentation on the pictures. Since our data is dog classification, we can do many augmentations. Of course be careful not to change the pictures excessively so that the dog will no longer be in the picture, or will be in a very vague way.

1.3 Is the data balanced

Labeled images class distribution -



An example of classes and the number of data they consists -
Class miniature_schnauzer samples: 3862
Class brittany_spaniel samples: 3928
Class kelpie samples: 4006
Class papillon samples: 4093
Class border_collie samples: 4158

We can see that the data also varied in representation of the samples in the dataset. In order to overcome this diversion we uses Stratisfied KFold instead of KFold for our final models

1.4 benchmark results for different methods used on this data

There are many different methods used on this data. Some of them used known models. for example some solutions describe above -
VGG - Loss = 0.352, Accuracy 0.918.
Xception - Loss = 0.068, Accuracy = 0.9809
Inception - Loss = 0.08069, Accuracy = 0.967

1.5 Samples from labels

The data consists of dog images, therefore there are no easily separable categories, and harder more similar categories. But, we can see that there are some images that are harder to classify than others, such as the ones that there are humans beside the dogs.

2. Form a neural network

2.1 5fold cross validation

We used the KFold cross validation technique with K = 5 in order to bla bla.

2.2(1) First model results analyze

The first model we ran in order to solve the problem is a simple CNN model.

```
Model: "functional_219"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_110 (InputLayer)       [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d_446 (Conv2D)          (None, 126, 126, 16)      448
_________________________________________________________________
max_pooling2d_424 (MaxPoolin (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_447 (Conv2D)          (None, 61, 61, 32)        4640
_________________________________________________________________
max_pooling2d_425 (MaxPoolin (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_448 (Conv2D)          (None, 28, 28, 48)        13872
_________________________________________________________________
max_pooling2d_426 (MaxPoolin (None, 14, 14, 48)        0
_________________________________________________________________
flatten_109 (Flatten)        (None, 9408)              0
_________________________________________________________________
dense_402 (Dense)            (None, 64)                602176
_________________________________________________________________
dense_403 (Dense)            (None, 120)               7800
=================================================================
Total params: 628,936
Trainable params: 628,936
Non-trainable params: 0
_________________________________________________________________
```

The optimizer we choose for the model is 'Adam'.
the results of the model are - \\ % results

We can see that our naive model approach did not work.
The model did not manage to memorize the data, and the validation results are almost entirely random. Lets suggest a few ways that can possibly improve that.

2.3(1) Suggested ways to improve towards second iteration

// explain why we think the model wasn't good

- Increase model complexity by adding 2 more convolutional layers.
- Add 2 more dense layer to increase parameters count to better identify features.
- Add batch normalization after each convultional layer to help coordinate the update of multiple layers in the model (leads to higher learning rate and better speed).

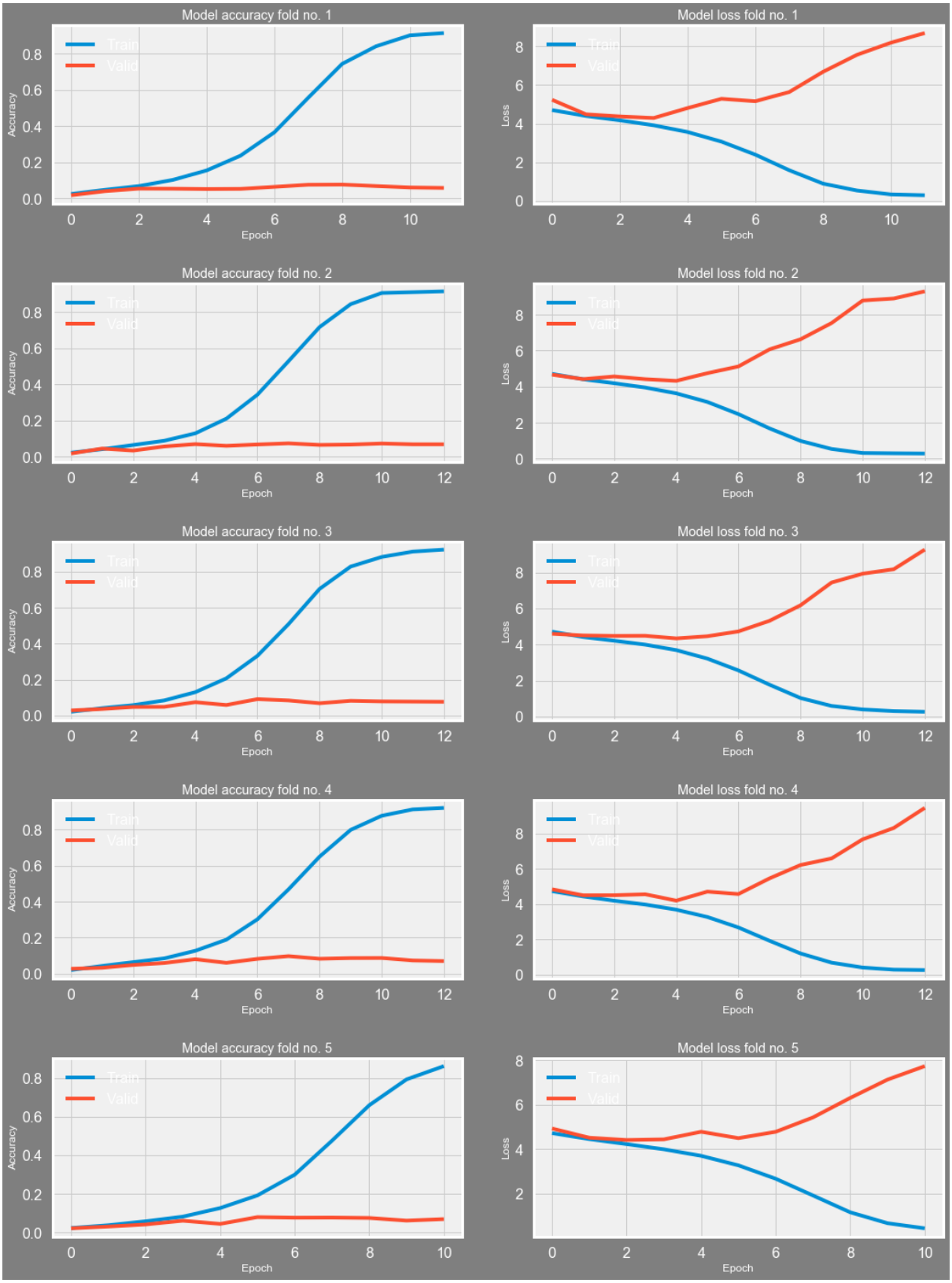
2.2(2) Second model results analyze

In our second model we tried to implement the 3 improvements we suggested in section 2.3(1). our new CNN model is -

```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 126, 126, 16)      448
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 126, 16)      64
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 61, 32)        4640
_________________________________________________________________
batch_normalization_1 (Batch (None, 61, 61, 32)        128
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 28, 48)        13872
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 28, 48)        192
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 48)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 64)        27712
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 12, 64)        256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
```

```
dense (Dense)                (None, 1024)              2360320
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_3 (Dense)              (None, 120)               30840
=================================================================
Total params: 3,094,600
Trainable params: 3,094,280
Non-trainable params: 320
_________________________________________________________________
```

In addition, we added batch normalization after each convolutional layer.
the results of the model are -
Mean fold accuracy: 90.73%
Mean fold loss: 31.3%
Mean fold validation accuracy: 6.87%
Mean fold validation loss: 889.21%
Kaggle multiclass loss score: 7.06920 (Lower is better)



We can see that we encounter overfitting problem since the train accuracy increases while the validation loss stops increasing at an early stage.

2.3(2) Suggested ways to improve towards second iteration
// explain why we think the overfitting

- Add dropout layers to avoid memorization of features in the model and to try and decrease overfitting problem
- Increase input shape ((128, 128, 3) => (224, 224, 3)) to better preserve original samples features
- Increase model complexity by adding convolutional layers to try and better represent sample features
- Use data augmentation to increase the sample count of training data and to better generalize the available training data
- Use stratisifed KFold to avoid diversions caused by uneven sample count per label and to better represent the label distirbution in each fold
- Deacrease optimizer learning rate ($1e^{-2}$ => $1e^{-3}$) to increase reliability in gradient decent convergence
- Increase epoch count to try and acheive covergence due to the decreased learning rate (20 => 40 epochs) and usage of dropout layers
- Normalize the samples pixel values to range [0, 1] to increase the models convergence speed in order to compensate the now decresed learning rate
- Add global average pooling layer before dense layer to try and pass average features to fully-connected layers

2.2(3) Third model results analyze

In our Third model we tried to implement the improvements we suggested in section 2.3(2). our new CNN model is -

```
Model: "functional_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_7 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 222, 222, 16)      448
_________________________________________________________________
batch_normalization_30 (Batc (None, 222, 222, 16)      64
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 111, 111, 16)      0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 109, 109, 32)      4640
_________________________________________________________________
batch_normalization_31 (Batc (None, 109, 109, 32)      128
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 54, 54, 32)        0
_________________________________________________________________
conv2d_32 (Conv2D)           (None, 52, 52, 64)        18496
_________________________________________________________________
batch_normalization_32 (Batc (None, 52, 52, 64)        256
_________________________________________________________________
max_pooling2d_28 (MaxPooling (None, 26, 26, 64)        0
_________________________________________________________________
conv2d_33 (Conv2D)           (None, 24, 24, 128)       73856
_________________________________________________________________
batch_normalization_33 (Batc (None, 24, 24, 128)       512
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_34 (Conv2D)           (None, 10, 10, 256)       295168
_________________________________________________________________
```

```
batch_normalization_34 (Batc (None, 10, 10, 256)       1024
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 5, 5, 256)         0
_________________________________________________________________
global_average_pooling2d_6 ( (None, 256)               0
_________________________________________________________________
flatten_6 (Flatten)          (None, 256)               0
_________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
_________________________________________________________________
dense_18 (Dense)             (None, 512)               131584
_________________________________________________________________
dropout_13 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_19 (Dense)             (None, 256)               131328
_________________________________________________________________
dense_20 (Dense)             (None, 120)               30840
=================================================================
Total params: 688,344
Trainable params: 687,352
Non-trainable params: 992
_________________________________________________________________
```

The results of the model are -

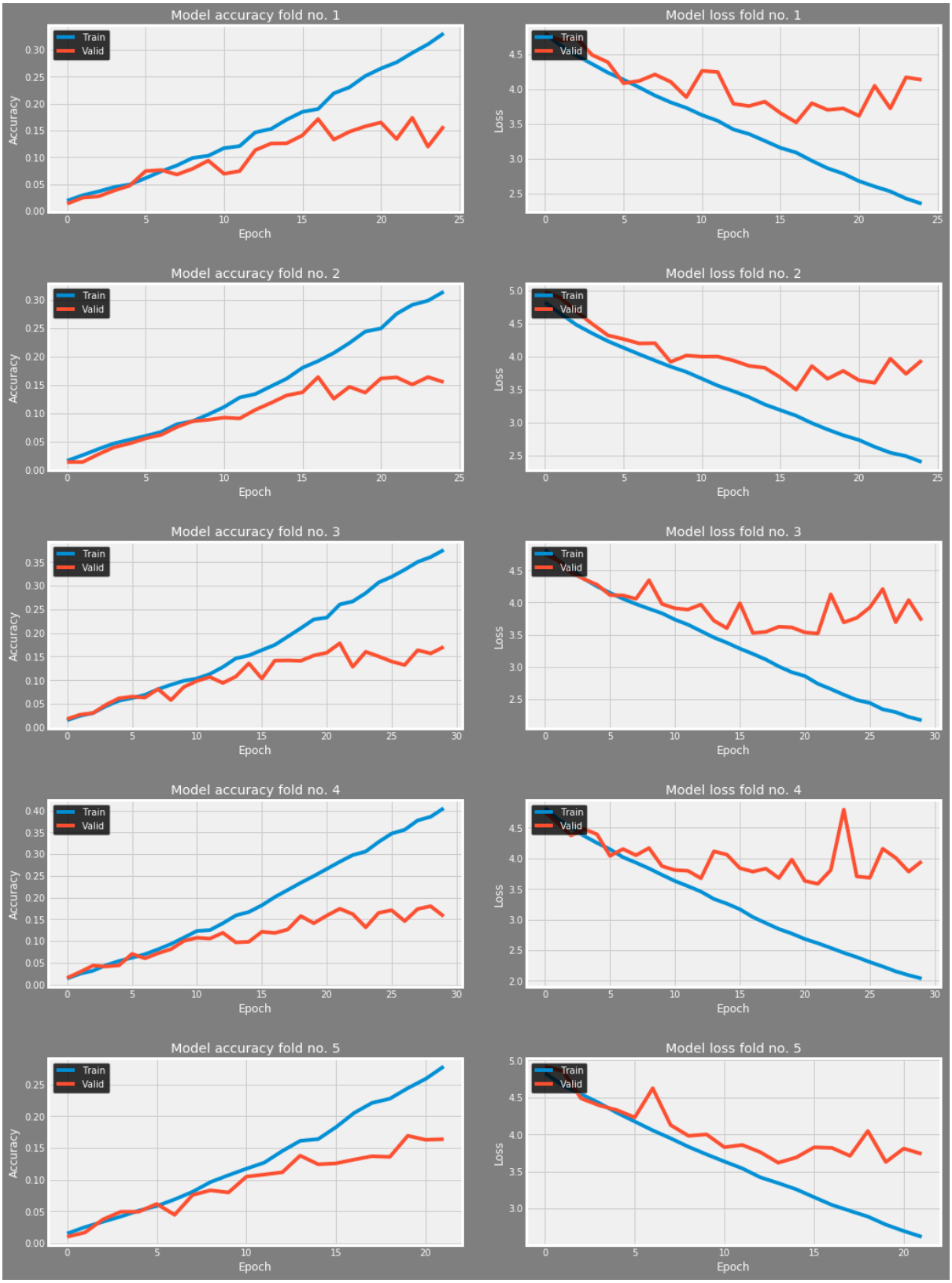
Mean fold accuracy: 34.07%

Mean fold loss: 2.3148712635040285

Mean fold validation accuracy: 16.07%

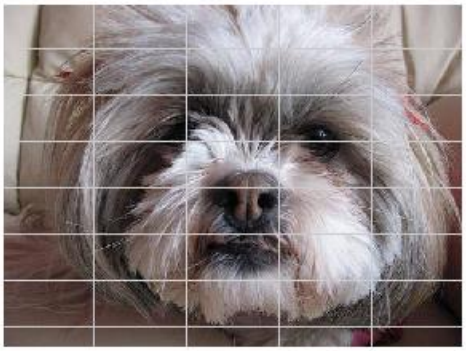
Mean fold validation loss: 3.8972976207733154

Kaggle multiclass loss score: 6.71512 (Lower is better)

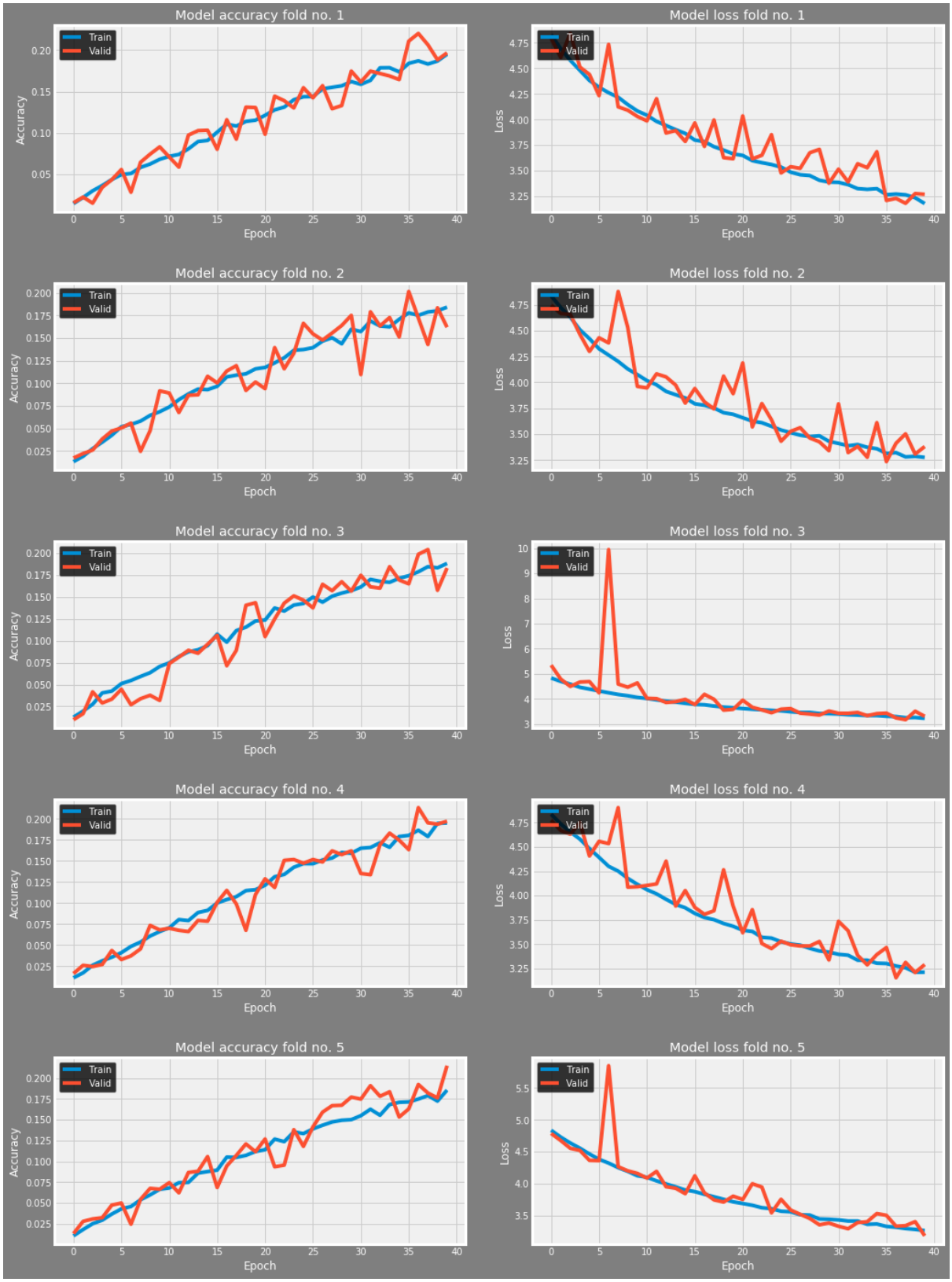

Model accuracy fold no. 1
Model loss fold no. 1
Model accuracy fold no. 2
Model loss fold no. 2
Model accuracy fold no. 3
Model loss fold no. 3
Model accuracy fold no. 4
Model loss fold no. 4
Model accuracy fold no. 5
Model loss fold no. 5
Train
Valid
Accuracy
Loss
Epoch

2.5 Inference Time Data Augmentation

We used data augmentation on the training data to increase total training samples and to better generalize the input.
augmentation sample -



The results -
Mean fold accuracy: 18.96%
Mean fold loss: 3.23105731010437
Mean fold validation accuracy: 19.07%
Mean fold validation loss: 3.285396909713745
Kaggle multiclass loss score: 5.11512 (Lower is better)



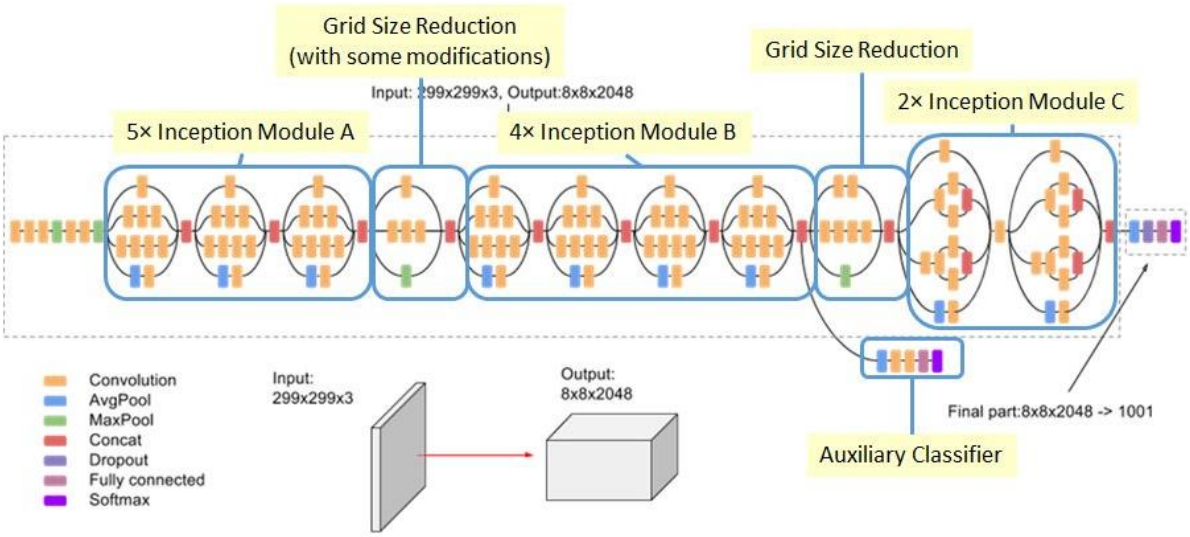
We can see that using data augmentation improved our overall results, possibly with more epochs of training this simple model will be able to lower the train/validation loss

3. Transfer learning using InceptionV3

In this section we are going to use transfer learning techniques to solve our problem using the pretrained known model InceptionV3
Inception v3 is a widely-used image recognition model that has been shown to attain greater than 78.1% accuracy on the ImageNet dataset.

The model is the culmination of many ideas developed by multiple researchers over the years. It is based on the original paper: "Rethinking the Inception Architecture for Computer Vision" by Szegedy, et. al.

A high-level diagram of the model is shown below:



3.1

The best results we achieved as can be seen from the summary table above were by using the InceptionV3 model with an input layer with dimensions (224, 224, 3) and a last Fully Connected classification block that consists of the following layers:

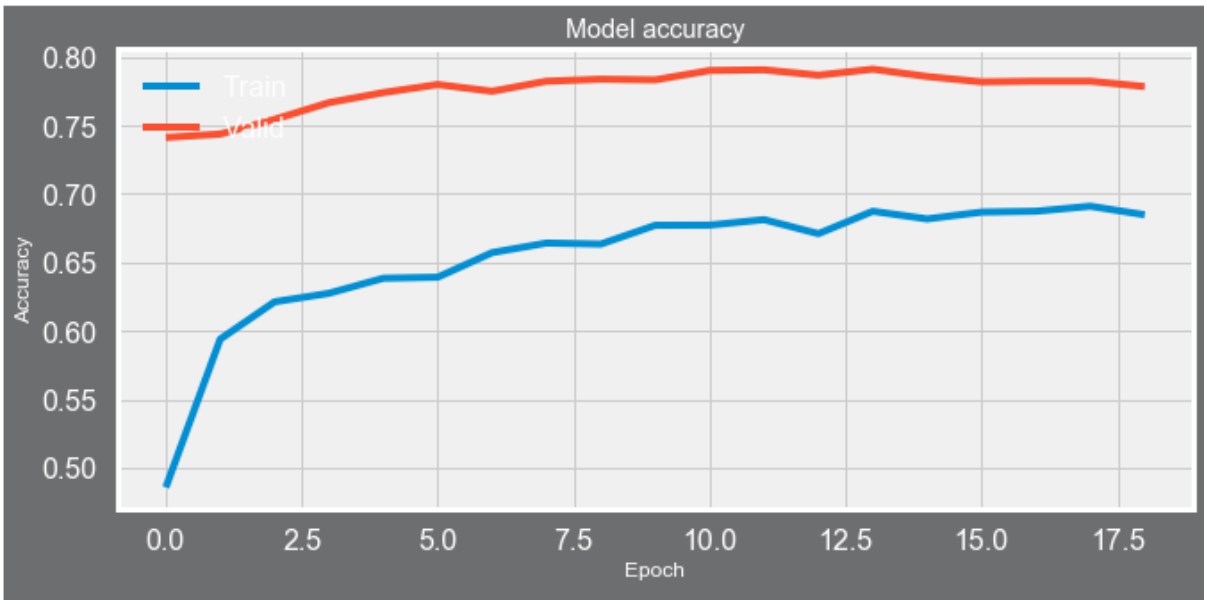
```
_____________________________________________________________________________________________
global_average_pooling2d_26 (Gl (None, 2048)         0         mixed10[0][0]
_____________________________________________________________________________________________
dense_78 (Dense)                (None, 512)          1049088   global_average_pooling2d_26[0][0]
_____________________________________________________________________________________________
batch_normalization_224 (BatchN (None, 512)          2048      dense_78[0][0]
_____________________________________________________________________________________________
dropout_52 (Dropout)            (None, 512)          0         batch_normalization_224[0][0]
_____________________________________________________________________________________________
dense_79 (Dense)                (None, 256)          131328    dropout_52[0][0]
_____________________________________________________________________________________________
batch_normalization_225 (BatchN (None, 256)          1024      dense_79[0][0]
_____________________________________________________________________________________________
predictions (Dense)             (None, 120)          30840     batch_normalization_225[0][0]
=============================================================================================
Total params: 23,017,112
Trainable params: 22,981,144
Non-trainable params: 35,968
```

3.2

The main pre-processing step we took was dividing the data set to stratified train - test split. In addition we performed one-hot encode on the output.

3.3

By the predictions samples we received that we got big improvement by using InceptionV3.
Most of the wrong predictions that we got are hard to seperate just by looking at the pictures(as can be seen in the output attached).
The wrong predictions dog breeds are very similar looking in terms of shape, hair, color and size.
We can also see that the correct and wrong predictions are very high for the top samples, this can indicate that our model is very decisive this time around.



3.4

We researched regarding the support vector machine classifier.

About Support Vector Machine (SVM)

Support-vector machines (SVMs, also support-vector networks) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis.

More formally, a support-vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks like outliers detection.

A good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin, the lower the generalization error of the classifier.

Using Linear Support Vector Machine algorithm

We used the model trained in the previous section to extract our dataset features and apply the Linear support vector machines (SVM) ML algorithm to classify our data using those features

3.5

During our research we ended having 5 iterations of a simple CNN model before using transfer learning with the InceptionV3 netwrok. Our results were as follows:

Results Summary

Model	Main Changes	Batch Size	Epochs	Runtime	Input Shape	Optimizer
Simple CNN Iteration 1	-	20	20	6min approx.	(128, 128, 3)	Adam
Simple CNN Iteration 2	- Added Convolutional layers with filter size 64 - Added two FC layers and increased neuron count (512, 1024) - Added Normalization	20	20	8min approx.	(128, 128, 3)	Adam
Simple CNN Improved	- Increased input size to (224, 224, 3) - Added Droupout layers (0.5) between FC layers - Added 2 Convolutional layers (128 and 256 filters) - Added Global average pooling between Conv and FC layers - Removed 1024 neurons FC layer - Decreased learning rate to 0.001 - Used Stratisfied KFold instead of KFold - Normalized input from [0, 255] to [0, 1]	20	40	21min approx.	(224, 224, 3)	Adam
Simple CNN Improved w/ Data augmentation	- Added inference time data augmentation using ImageDataGenerator	20	40	1hr approx.	(224, 224, 3)	Adam
Simple CNN Improved w/ Data augmentation - Second train	- Trained previous model again for 20 more epochs	20	40 + 20	1hr + 30min approx.	(224, 224, 3)	Adam
Transfer Learning using InceptionV3	- Used transfer learning with the InceptionV3 model - Changed input dims to (224, 224, 3) - Removed classification layer and added the following: * Global average pooling * FC 512 filters * Batch Normalization * Droupout 0.5 * FC 256 filters * Batch Normalization * FC classification layer softmax activation(120 labels)	20	20	46min approx.	(224, 224, 3)	Adam
						Kernel BD
IncaeptionV3 model w/ Feature Extraction using SVM	- Extracted features from previously trained InceptionV3 based model by predicting on train set. - Classified using Support Vector Machine ML algorithm	20	40 /w GridSearchCV hyperparameter tuning	40min approx.	Train set dims 8022 approx.	RBF

Model	Learning Rate	Train Accuracy	Train Loss	Validation Accuracy	Validation Loss	Loss Function	Kaggle Multiclass Loss Score
Simple CNN Iteration 1	0.01 (default)	1.2%	4.7	0.8%	4.8	Categorical Crossentropy	-
Simple CNN Iteration 2	0.01 (default)	90.73%	3.1	6.87%	8.89	Categorical Crossentropy	7.06920
Simple CNN Improved	0.001	34.07%	2.31	16.07%	3.89	Categorical Crossentropy	6.61315
Simple CNN Improved w/ Data augmentation	0.001	18.96%	3.23	19.07%	3.2	Categorical Crossentropy	4.91512
Simple CNN Improved w/ Data augmentation - Second train	0.001	23.22%	2.9	29.2%	2.8	Categorical Crossentropy	4.11512
Transfer Learning using InceptionV3	0.001	68.9%	1.07	78.2%	0.72	Categorical Crossentropy	1.85799
	Gamma						
IncaeptionV3 model w/ Feature Extraction using SVM	0.001 0.0001	43%	-	32%	-	-	-

4. Summary

MAIN EMPHASIZE
During the assignment we used variation of techniques in order to produce better results and improve our model gradually.
We researched and realized the following techniques and methods:

- Using KFold cross validation
- Using Stratisfied KFold cross validation
- Tuning model hyperparatmetrs to achieve better results
- Overcoming Overfitting problem
- Using Transfer learning with a known network to produce better results
- Using SVM classifier on an exisiting CNN model features extracted from a test set

CONCLUSIONS

- During the training process we encountered overfitting in two of the simple model iterations. We found out that the method that helped reduce the overfitting problem the best was using Dropout layers. In our third model iteration we used two Dropout layers between the last two Fully connected layers with probability of 0.5. Using this method, we were able to eliminate overfitting but achieved underwhelming results and out training time almost tripled in relative to the training loss drop.
- Another thing that we found out to cause overfitting is having too many parameters coming from the Fully Connected layers. A thing that with the lack of dropout caused our models to memorize the training data very quickly but came up will almost no results when it comes to the validation data.
- Decreasing the learning rate (With the Adam optimizer) also had a big difference one we added the dropout layers. The learning process was slower, but our validation loss kept in line with the training loss in almost all cases.
- We noticed that when adding dropout, our validation accuracy was sometimes higher than the training accuracy. Our explanation for it was that due to disabling neurons, some of information about each sample is lost, and the subsequent layers attempt to construct the answers basing on incomplete representations. The training loss was higher because we've probably made it artificially harder for the network to give the right answers. However, during validation all the neurons are available, so the network has its full computational power - and thus it might perform better than in training.