

[Return to Classroom](#)[DISCUSS ON STUDENT HUB](#)

Dog Breed Classifier

REVIEW

HISTORY

Meets Specifications

Congratulations! You've passed the project!

1. You did a great job by trying and testing out different things. We should always remember, we learn only when we experiment a lot and make mistakes :)
2. Dog breed classifier [report](#) by a Stanford student.
3. As you've already got familiarity with image classification, I'll recommend reading about [object detection](#) as well.
4. Fun fact: In the real world we rarely train models from scratch. The already pre-trained models provide a good initialization for fine-tuning according to our tasks. Pre-trained weights are even used as weight initializers for training the whole network!

Good luck with your future projects!

Files Submitted



The submission includes all required, complete notebook files.

Both the report file and the jupyter notebook submitted.

Step 1: Detect Humans



The submission returns the percentage of the first 100 images in the dog and human face datasets that include a detected, human face.

You've correctly detected the percentage of human faces in human and dog datasets

Step 2: Detect Dogs



Use a pre-trained VGG16 Net to find the predicted class for a given image. Use this to complete a `dog_detector` function below that returns True if a dog is detected in an image (and False if not).

Using "return predict >= 151 and predict <= 268" is clever programming. No use of if or else statements. Great work!



The submission returns the percentage of the first 100 images in the dog and human face datasets that include a detected dog.

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)



Write three separate data loaders for the training, validation, and test datasets of dog images. These images should be pre-processed to be of the correct size.

1. Some transforms that can be used for training images are
transforms.RandomResizedCrop(),
transforms.RandomRotation(),
transforms.RandomHorizontalFlip()
2. Shuffling required for training data, no shuffling required for validation and test. An article on randomness you may want to [read](#).



Answer describes how the images were pre-processed and/or augmented.

1. You've correctly described the pre-processing step involving resizing, center-cropping, and augmentation.
2. Augmenting the dataset prevents overfitting as it introduces more variety to the dataset in addition to increasing the dataset size. More on overfitting [here](#).



The submission specifies a CNN architecture.

1. Using at least 3 convolutional layers helps the model to be complex enough to accurately recognize dog breeds. Well done!
2. Pooling is a great way to reduce network parameters. It also introduces invariance in the model. You can read about uses of pooling and different pooling layers [here](#).
3. Tip: Over the years, the research community has also tried an all convolutional network. Suggested [paper](#).
4. You've used dropout which is an incredible technique to prevent overfitting of networks. Tip: Use dropout = 0.5 as that's what works well for everyone unless you have an experimental analysis or an intuitive reason for taking some other value.
5. Batch normalization can be used in CNNs to help deal with internal covariate shift. A useful [article](#).
6. Using ReLu is a good idea. The post [here](#) details some other activation functions used in neural networks.



Answer describes the reasoning behind the selection of layer types.

1. You have provided sound reasoning for your network. You may also want to test what happens if you use aggressive downsampling.
2. The more you experiment, the more understanding you will have of how and why the network works the way it does :)



Choose appropriate loss and optimization functions for this classification task. Train the model for a number of epochs and save the "best" result.

1. You've used SGD which is one of the optimizers used by the deep learning community. You should definitely read about other optimizers [here](#).
2. Another good post on optimizers [here](#).



The trained model attains at least 10% accuracy on the test set.

21% is a good enough accuracy to pass the project. Students who experimented a lot have even got as much as 25% accuracy!

Step 4: Create a CNN Using Transfer Learning



The submission specifies a model architecture that uses part of a pre-trained model.

1. You have used VGG net. Good work! Other possible networks are ResNet variations, InceptionNet variations. Do try them as well if you have some time.
2. Tip: Adding one or two fully connected layers of your own after the final layer of the network and before the classification layer can improve the results you get.



The submission details why the chosen architecture is suitable for this classification task.



Train your model for a number of epochs and save the result with the lowest validation loss.

Not much training required as we are using a pre-trained model here



Accuracy on the test set is 60% or greater.

Well done!



The submission includes a function that takes a file path to an image as input and returns the dog breed that is predicted by the CNN.

Good work! The function should take a file path as argument and return the predicted dog breed.

Step 5: Write Your Algorithm



The submission uses the CNN from the previous step to detect dog breed. The submission has different output for each detected image type (dog, human, other) and provides either predicted actual (or resembling) dog breed.

Step 6: Test Your Algorithm



The submission tests at least 6 images, including at least two human and two dog images.

It's great that you tested your model on different images.



Submission provides at least three possible points of improvement for the classification algorithm.

1. Well analyzed! The listed points on improvement are indeed practical.
2. Tip: You may want to read 'Ensembles of models'. More on it [here](#)
3. Also, more augmentation techniques, larger ConvNets along with regularization techniques give better performing models.

 [DOWNLOAD PROJECT](#)

[RETURN TO PATH](#)