Spring 2025 STAT 742 - Advanced Machine Learning Final Project - Stanford Dogs Dataset

Daniel Tiourine

May 20, 2025

Contents

1 Introduction

In this report, we will explore and train several neural networks on the Stanford Dogs Dataset. This dataset was created by Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei at Stanford University as part of their research in fine-grained visual categorization. It contains images of 120 breeds of dogs from around the world, making it a challenging benchmark for image classification algorithms.

It consists of 20,580 images with 150-200 images per dog breed, all sourced from ImageNet. The dataset presents several challenges that make it ideal for evaluating different neural network architectures. The high intra-class variation (dogs of the same breed can look quite different) and inter-class similarity (different breeds can look remarkably similar) create a difficult classification problem that requires models to learn subtle distinguishing features.

The analysis will involve implementing and evaluate three distinct neural network approaches with various complexity: (1) a transfer learning approach utilizing a pre-trained ResNet50 model, (2) a simple artificial neural network using only fully-connected layers, and (3) a custom convolutional neural network trained from scratch. For each architecture, I analyze training dynamics, convergence patterns, and final performance metrics on validation and test sets.

The principal results of the analysis show a clear difference in performance, with the transfer learning approach achieving 81% test accuracy, dramatically outperforming both the custom CNN (51% accuracy) and the simple ANN (merely 6% accuracy). This substantial performance gap highlights two critical insights in deep learning for computer vision: the importance of appropriate architectural design (convolutional vs. fully-connected) and the advantage of knowledge transfer from pre-trained models when working with challenging datasets.

2 Data

2.1 Data Overview

Let's dive more deeply into the details of the Stanford Dogs Dataset. Some of the specific dog breeds represented in the dataset include: Afghan Hound: 199 images; African Wild Dog: 168 images; Airedale Terrier: 181 images; American Staffordshire Terrier: 183 images; Appenzeller Sennenhund: 143 images; Australian Terrier: 152 images; Basenji: 168 images; Basset Hound: 155 images; Beagle: 174 images; Bedlington Terrier: 142 images; Bernese Mountain Dog: 180 images; Blenheim Spaniel: 159 images; Bloodhound: 134 images, as well as many more. For example, below is an image of a Chihuahua from the dataset.

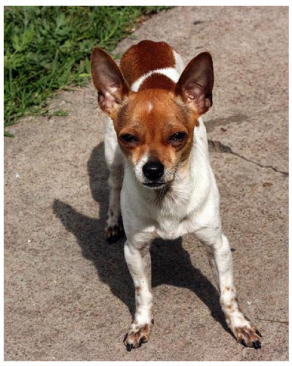

Figure 1: Sample image of a Chihuahua from the Stanford Dogs Dataset.

All images in the dataset feature at least one dog as the primary subject, though the images vary significantly in terms of dog pose, size within the frame, lighting conditions, and background complexity. Each image is annotated with a bounding box specifying the position of the dog, though in the analysis I did not use this information and instead used the full images rather than the cropped regions. The images were collected "in the wild," meaning they represent real-world scenarios with natural variation in photographic quality, dog positioning, and environmental context.

The dataset is structured hierarchically with breed-specific folders containing the corresponding images. Each image is labeled with a unique identifier that includes the breed code and an image number (e.g., "$n02085620_7.jpg$" for a Chihuahua image). The breed codes follow the WordNet IDs from ImageNet, providing standardized categorization.

For the analysis, I split the dataset into training (70%), validation (15%), and testing (15%) sets using `scikit-learn` to maintain the class distribution across splits. This division ensures approximately 140 images per breed in the training set, with 30 images each in the validation and test sets, providing sufficient data for both model training and reliable performance evaluation.

2.2 Data Preprocessing

Data preprocessing plays an important role in preparing the dataset for effective neural network training. For each model in this analysis, the preprocessing steps had to be done slightly differently, but some fundamental techniques apply across all models.

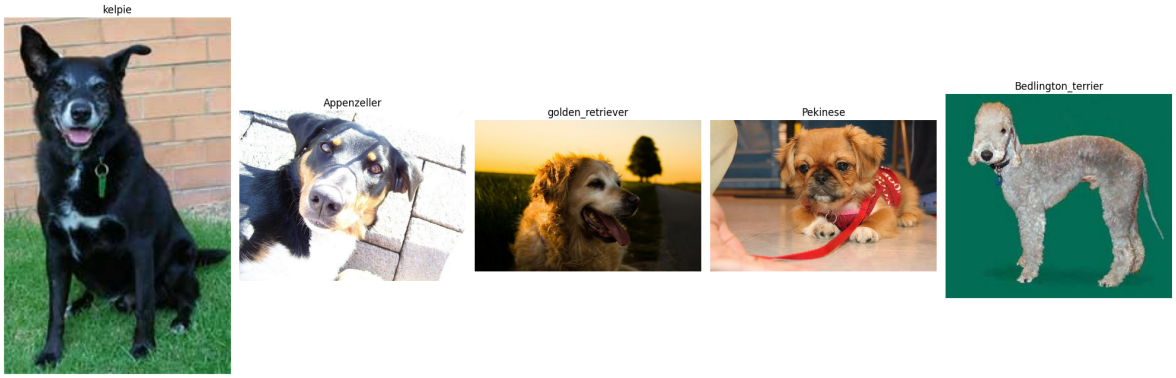

Figure 2: Sample images from the Stanford Dogs Dataset.

First all the images in the dataset were resized to a consistent dimension of 224 x 224 pixels with three color channels (RGB). By having uniform dimensions across the images, the network will be able to process them in batches, which significantly accelerates training. Additionally, the 224 x 224 dimension specifically matches the input requirements of ResNet50 and other architectures, facilitating transfer learning.

Next, the images were normalized by scaling the pixel values from the original range [0, 255] to [0, 1] by dividing by 255. This normalization helps stabilize the gradient descent proccess and enable faster convergence by placing all pixel values on a similar scale.

For ResNet50 specifically, a specialized preprocessing function `preprocess_input()` was necessary to ensure the input data matched the distribution of the data used during the model's original training, which is crucial for effective transfer learning.

2.3 Data Augmentation

Data augmentation is a crucial technique in deep learning that artificially expands the training dataset by creating modified versions of existing images. This process is essential for improving model robustness and generalization as it helps prevent the model from memorizing specific examples rather than learning meaningful features.

To improve model generalization and combat overfitting, I implemented data augmentation on the training set. These augmentations included:

- **Random horizontal flipping:** Creates mirror images, helping the model learn that left-right orientation doesn't change the breed.
- **Random brightness adjustments:** Varies image brightness by up to $\pm 20\%$, simulating different lighting conditions.
- **Random contrast modifications:** Alters image contrast by a factor between 0.8 and 1.2, mimicking different photographic qualities.
- **Random saturation changes:** Modifies color intensity to help the model focus on shape and pattern rather than specific color tones.

These augmentations increase the effective size of the training dataset, expose the models to greater variation, and improve robustness to common image variations. Importantly, augmentation was only applied to the training data—validation and test sets remained unaugmented to accurately assess model performance on real-world examples.

3 Analysis

3.1 Overview

In the analysis, I trained several neural networks on the Stanford Dogs Dataset to compare their performance on this challenging image classification task. The models range from a simple artificial neural network (ANN) consisting only of fully-connected layers to a transfer learning approach using a pre-trained ResNet model. I also implemented a custom convolutional neural network (CNN) trained from scratch to compare them to the above two approaches.

3.2 Transfer Learning

The first model was trained with transfer learning. Transfer learning is an important deep learning technique where we allow models to leverage knowledge gained from one task to improve performance on another. This approach is particularly valuable in image classification, where low-level visual features like edges, textures, and patterns are common across many different classification domains.

Rather than training a neural network from scratch with randomly initialized weights, transfer learning begins with a model pre-trained on a large dataset. This pre-training enables the network to develop a rich hierarchy of visual features: early layers capture basic elements like edges and textures while later layers capture more complex patterns and shapes.

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import  GlobalAveragePooling2D, Dense, Dropout,
    BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.regularizers import l2

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(
    img_height, img_width, 3))

# Freeze the base model layers
for layer in base_model.layers:
    layer.trainable = False

# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(640, activation='relu', kernel_regularizer=l2(0.0015))(x)
x = BatchNormalization()(x)
x = Dropout(0.65)(x)
x = Dense(320, activation='relu', kernel_regularizer=l2(0.0015))(x)
x = BatchNormalization()(x)
x = Dropout(0.65)(x)
predictions = Dense(num_classes, activation='softmax')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

The pretrained model I selected was ResNet50 (a residual network with 50 layers). The transfer learning process first involves freezing all layers in the base model by setting the `trainable` parameter to `False`, ensuring that the valuable pre-trained features would remain fixed during initial training. I also had to add a custom classification head so the output of the neural network matched the number of classes in the dataset.

To train the pretrained model with weights frozen, I used the Adam optimizer with a learning rate of 0.0002. Since the task involved classifying amount multiple classes, the sparse categorical crossentropy function was chosen. For the same reason, the neural network used a softmax layer for output.

After training with frozen weights for 20 epochs, I then fine-tuned the model on the Stanford Dogs dataset. Fine-tuning is a critical second stage in transfer learning where selected layers of the pre-trained network are unfrozen and allowed to adapt to the specific characteristics of the target dataset. For this phase, I carefully unfroze the last 20 layers of the ResNet50 base model while keeping earlier layers fixed, as these later layers contain more dataset-specific features that benefit from adaptation to the dog breed classification task.

After this whole training and finetuning process, the network performed well on the Stanford Dogs Dataset, achieving 90% training accuracy, 82% validation accuracy, and 81% test accuracy.

3.3 Simple Feedforward Neural Network

The next model used in this analysis was a simple feed-forward Artifical Neural Network alongside the convolutional and transfer learning approaches. This traditional multilayer perceptron architecture serves as an important baseline, allowing us to quantify the benefits of specialized architectures for image classification tasks.

```
ann_model = models.Sequential([
    layers.Flatten(input_shape=input_shape),
    layers.BatchNormalization(),

    layers.Dense(512, activation='elu', kernel_initializer='he_normal'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(384, activation='elu', kernel_initializer='he_normal'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(256, activation='elu', kernel_initializer='he_normal'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(128, activation='elu', kernel_initializer='he_normal'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),

    layers.Dense(num_classes, activation='softmax')
])
```

The ANN model fundamentally differs from convolutional approaches in its treatment of input data. It first flattens each 224 x 224 x 3 image into a one-dimensional vector of 150,528 values. This discards the spatial structure of the data, treating pixels from opposite corners of the image no differently than adjacent pixels.

After flattening, the network consists of several fully-connected layers with decreasing sizes (512, 384, 256, 128 neurons), each followed by batch normalization and dropout for regularization, before the final softmax classification layer. Despite its simplicity, the architecture actually contains substantially more parameters than the CNN model due to the massive number of connections from the flattened input layer to the first hidden layer. CNNs, on the other hand, have the property of parameter sharing, which means the same filter weights are applied across the entire image. For example, if a CNN has a 3×3 filter that detects edges, those exact same 9 weights are reused as the filter slides across the image, rather than learning separate weights for each position as a fully-connected network would. This dramatically reduces the number of parameters while maintaining the ability to detect features regardless of their position in the image, making CNNs both more efficient and better suited for image processing tasks.

Due to these challenges, it would be expected for the ANN to perform much worse on the data.

After compiling the model with Nesterov accelerated stochastic gradient descent with a learning rate of 0.01 and momentum of 0.9 and training on 50 epochs, the model achieved an accuracy of about 6% on the test set. While this is very poor performance, this is expected for a simple network on a challenging image dataset like the Stanford Dogs Dataset. While fine-tuning the hyperparameters could help performance a little bit, the issue mainly stems from the fundamental limitations of the model's architecture. Without the spatial feature detection capabilities and hierarchical representation learning that CNNs provide, the basic ANN lacks the appropriate capability needed to effectively process and classify complex image data.

3.4 Convolutional Neural Network

The third model used in the analysis is a Convolutional Neural Network without transfer learning. This serves as an intermediate point between the simple ANN and the transfer learning approach, allowing us to isolate the benefits of the CNN architecture itself from the advantages provided by pre-trained weights. Unlike the ANN, this model incorporates convolutional layers that can detect spatial features in the images through learned filters, pooling operations that provide translation invariance, and the parameter sharing mechanism that dramatically reduces the number of weights compared to fully-connected networks.

However, unlike the transfer learning approach, this CNN must learn all its weights from scratch using only the training dataset. By implementing this model, we can better understand the relative contributions of both architectural design choices (CNN vs. ANN) and knowledge transfer (pre-trained vs. trained from scratch) to the overall performance on the challenging Stanford Dogs Dataset.

```
cnn_model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(
        img_height, img_width, 3),
                  kernel_regularizer=regularizers.l2(0.0002)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.3),

    layers.Conv2D(64, (3, 3), activation='relu', padding='same',
                  kernel_regularizer=regularizers.l2(0.0002)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.4),

    layers.Conv2D(128, (3, 3), activation='relu', padding='same',
                  kernel_regularizer=regularizers.l2(0.0002)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2, 2)),
    layers.Dropout(0.5),

    layers.Flatten(),
    layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2
        (0.0002)),
    layers.BatchNormalization(),
    layers.Dropout(0.6),
    layers.Dense(num_classes, activation='softmax')
])
```

The network begins with multiple convolutional blocks, each following a similar pattern of increasing complexity. The first block contains two Conv2D layers with 32 filters of size 3×3 using ReLU activation and 'same' padding, each followed by batch normalization to stabilize training. A max pooling layer with a 2×2 window then downsamples the feature maps, and a dropout layer with progressively increasing rates reduces overfitting. This pattern repeats in subsequent blocks, with the number of filters doubling each time to capture increasingly abstract features as the spatial dimensions decrease.

After the three convolutional blocks, the network flattens the extracted features and passes them through a dense layer of neurons, batch normalization, and a higher dropout rate of 0.6 before the final classification layer with softmax activation matching the number of dog breeds. The model was compiled using the Adam optimizer with a learning rate of 0.0005 and categorical cross-entropy loss function, like the other models. Training was performed with a maximum of 100 epochs.

The CNN model achieved a training accuracy of about 58% (it was able to achieve over 90% before I simplified the model to avoid overfitting on the validation set), and about 50% validation accuracy. This performance, while substantially better than the ANN's 6% accuracy, still falls short of the transfer learning approach. This aligns with what we would expect: the CNN architecture is better suited for image data, so it makes sense that it would perform much better than a simple ANN. However, the gap between the CNN and transfer learning model shows the value of knowledge transfer from pre-trained networks. The transfer learning approach benefits from millions of parameters already optimized on massive datasets, incorporating robust, general-purpose visual representations that is very helpful for other tasks.

4 Plots and Tables

The two plots below show the training and validation loss and accuracy for the transfer learning model discussed in section 3.2.

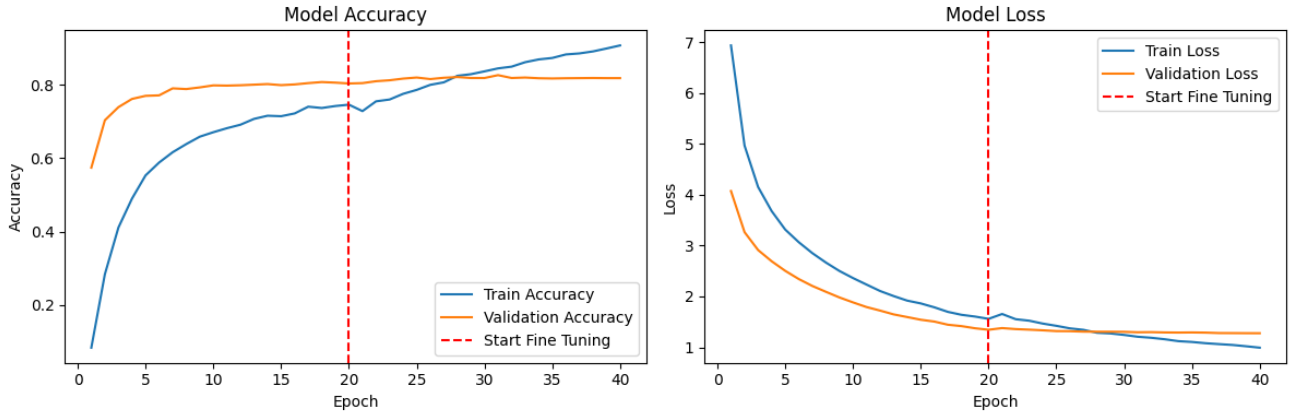

Figure 3: Training and validation metrics for the transfer learning model. Accuracy and loss curves show convergence and strong performance on the dataset with some slight overfitting.

Figure 3 presents the training loss and accuracy curve of the ResNet-based transfer learning model, demonstrating good performance that vastly outperforms both the ANN and CNN architectures. The accuracy plot reveals rapid convergence, with validation accuracy reaching approximately 78% within just 5 epochs, and stabilizing around 82% thereafter—a substantial improvement over the CNN's 50% validation accuracy.

This dramatic performance gap highlights the power of transfer learning, where the pre-trained ResNet model leverages knowledge extracted from millions of images on ImageNet. Notably, the fine-tuning phase, indicated by the vertical red dashed line at epoch 20, shows an interesting pattern: prior to fine-tuning, validation accuracy actually exceeded training accuracy, indicating the model was generalizing extremely well with the pre-trained feature extractors. After fine-tuning begins, training accuracy gradually surpasses validation accuracy, eventually reaching approximately 90% while validation accuracy remains stable around 82%. The loss curves confirm this pattern, showing steady convergence for both metrics with minimal divergence.

This relatively small gap between final training and validation performance (approximately 8%) despite the model's high capacity demonstrates the effectiveness of transfer learning in mitigating overfitting on challenging datasets.

The two plots below show the training and validation loss and accuracy for the simple feedforward neural network model discussed in section 3.3.

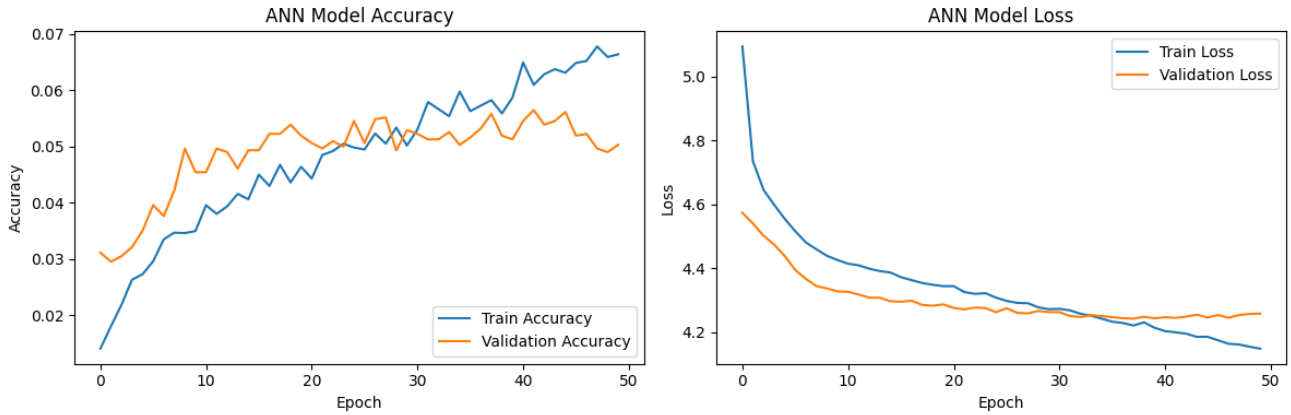

Figure 4: Training and validation metrics for the ANN model. Left: accuracy curves showing poor overall performance. Right: loss curves indicating incomplete convergence.

Figure 4 illustrates the training dynamics of the ANN model, revealing several limitations. The accuracy plot shows that even after 50 epochs, the model achieved extremely poor performance, with training accuracy peaking at roughly 6.7% and validation accuracy stagnating around 5%. This indicates the model's inability to capture meaningful patterns in the image data.

Additionally, the oscillating pattern in both training and validation accuracy curves suggests instability in the learning process, while the divergence between training and validation accuracy after epoch 30 indicates some overfitting. The loss curves confirm these observations, showing an initial rapid decrease followed by a slow, inconsistent convergence where validation loss eventually plateaus and slightly increases. These training plots demonstrate the inherent limitations of fully-connected architectures for complex image classification tasks, as they lack the spatial feature detection capabilities necessary to distinguish subtle visual patterns that differentiate dog breeds.

The two plots below show the training and validation loss and accuracy for the convolutional neural network model discussed in section 3.4

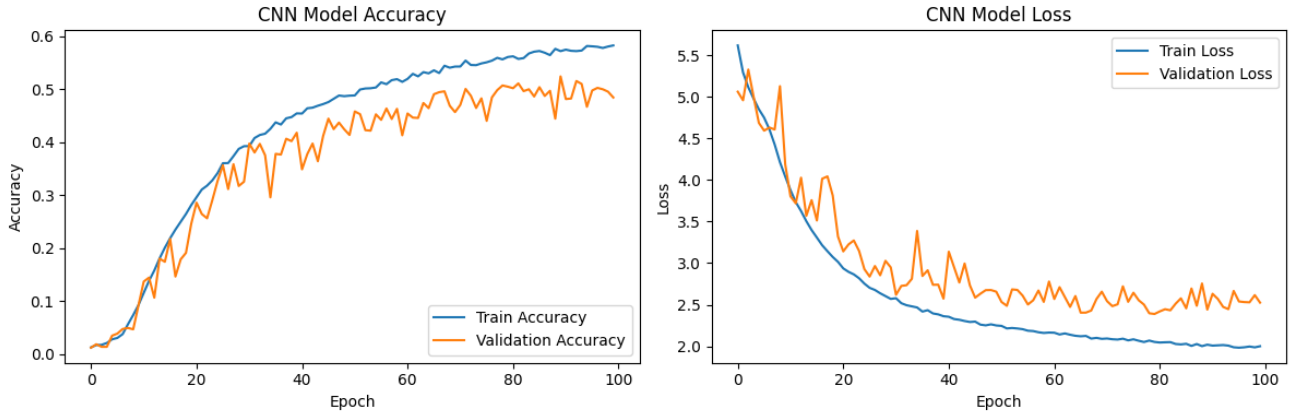

Figure 5: Training and validation metrics for the CNN model. Accuracy and loss curves showing oscillation and slight overfitting.

Figure 5 illustrates the training curve for the CNN model, revealing substantial improvements over

the ANN architecture. The accuracy plot shows impressive performance gains, with training accuracy steadily increasing to approximately 59% and validation accuracy reaching around 50% after 100 epochs. This represents a dramatic improvement compared to the ANN.

Additionally, the CNN's convergence pattern demonstrates more stable learning, with both metrics showing consistent upward trends through the first 60 epochs before the validation accuracy begins to plateau with some oscillation. The loss curves are consistent with this pattern, with both training and validation loss decreasing steadily and maintaining a smaller gap than observed in the ANN model.

While there is still evidence of moderate overfitting (approximately 9% gap between training and validation accuracy), the model has clearly learned meaningful representations of the dog breed features. With further training and fine-tuning the hyperparameters, it would likely be able to perform significantly better.

The oscillation in validation accuracy during later epochs suggests the model is approaching its capacity limit given the constraints of training from scratch on this challenging dataset. Overall though, these results confirm the advantages of convolutional architectures for image classification tasks, demonstrating the CNN's superior ability to learn spatial hierarchies of features through operations like convolution, pooling, and parameter sharing.

5 Conclusion

In conclusion, this report examined three distinct neural network approaches to classify the challenging Stanford Dogs Dataset, each revealing important insights about architectural design choices and transfer learning. The ANN model, which was a naive and simple model, performed poorly with only 6% accuracy, highlighting the fundamental limitations of fully-connected architectures for complex image recognition tasks.

The custom CNN demonstrated a marked improvement, achieving roughly 50% validation accuracy and 51% test accuracy by leveraging convolutional operations, parameter sharing, and hierarchical feature extraction capabilities specifically suited for image processing.

The transfer learning approach performed the best, capitalizing on its pre-trained weights. It was able to achieve roughly 90% training accuracy, 82% validation accuracy, and 81% test accuracy.

Comparing the performance of these different models and approaches demonstrates how the architecture of your model can make significant changes in the performance on the data. For example, we see how the architectural design (CNN vs. ANN) played a big difference. The CNN was able to significantly outperform the ANN, likely due to its parameter sharing capability and its ability to better capture visual features through a filter.

Additionally, knowledge transfer from pre-trained models offered a further advantage. The pre-trained base model was already able to capture low-level features like edges and other patterns, making it particularly well-suited for image classification tasks like distinguishing between similar dog breeds.

These findings emphasize that for complex image classification problems like dog breed identification, leveraging existing knowledge through transfer learning is not only beneficial for speeding up training but also very helpful for achieving the best performance.

6 Code

The code used for this analysis can be found on GitHub: https://github.com/dtiourine/stat-742-final-project