why?

Because sketches dont have clear lines and textures plus they can be complex and different from one another because of different drawing styles , no two people will draw the same exact sketch.

As you can see here these are two different sketches of a dog from the same dataset . You can see the massive difference between the styles of the two sketches . obviously the second sketch is much better and detailed more than the first sketch .

Also here we have two sketches for a bicycle from the same dataset as the dog sketches . you can also see here the obvious difference in the drawing style here the first one is very detailed for example the wheel has rims and the pedal is well drawn overall you could say this is a better sketch for a bicycle than the second one which has no details at all.

so as you can see sketch drawings are very diverse.

So in todays presentation , The goal is to provide

designers and artists with a tool for the creative process that can find related images

based on their sketches , and so by comparing 2 approaches first the tradiontal machine learning approach and secondly the transfer learning approach we can analyze the results and see who is more suited for our scenario.

First lets quickly discuss the diffrences between Tradional ML and Transfer Learning :

Traditional Machine Learning:

* Requires a large amount of labeled data to train a model from scratch.
* The model learns from scratch on the specific task it is trained on.
* The training process is usually time-consuming and computationally intensive.
* Once trained, the model can only perform well on the specific task it was trained on.

Transfer Learning:

* Utilizes a pre-trained model on a related task, allowing for better performance with less labeled data.
* The pre-trained model can be fine-tuned on a new task with fewer labeled examples. Which is very important in our case , we will see when we introduce the dataset.
* The fine-tuning process is usually faster and less computationally intensive than training a model from scratch.
* The pre-trained model can learn general features and patterns that can be transferred to new tasks, allowing for better performance on a wide range of tasks.

Tradional Machine Learning is Isolated single task learning : knowledge is not retained or accumulated . learning is performed without considering past learned knowledge in other tasks , on the other hand transfer learning , is learning of a new task relies on the previous learned tasks : learning process can be faster , more accurate and/or need less training data .

Ok so now lets introduce our dataset . The TU-Berlin dataset is a large-scale collection of sketches

from various object categories.

So why choose this dataset?

First of all , TU-berlin sketch dataset has over 20,000 sketches across 250 categories . Second sketches in the dataset are from a wide range of sources, including professional artists , novice sketchers and non-experts .Finally , the variety of sketch styles and skill levels enables models trained on this dataset to be more generalizable to new and unseen sketches.

Due to the limited computational power of my PC, I decided to select only 13 categories from

the TU-Berlin Sketch dataset for my project. These 13 categories were chosen based on their

diversity and prevalence in the dataset. I then divided the selected categories into training and

test sets, with 80% of the sketches being used for training and 20% for testing. This was done to

ensure that the model had sufficient data for training while still allowing for a reasonable amount

of testing to be conducted. Despite the reduced number of categories, I am confident that the

results obtained from this project will still be useful in providing insights into the performance

of the model and its ability to classify sketches accurately. Each Category of the 13 contain 80

sketches (60 for training and 20 for validation).

The Categories chosen were airplane apple banana bicycle car dog door ladder moon sheep table tree wheel.

Now for the first approach : Transfer Learning we have our first model VGG-16.

DataPRE

Before training the VGG16 model, we preprocessed the dataset by resizing all images to 224x224 pixels and normalizing the pixel values to be in the range [0, 1]. We also applied data augmenta-

tion techniques to the training set, including zooming with a factor of 0.15 , Width shifting with a factor of 0.2 , Height Shifting with a factor of 0.2 , and shearing with a factor of 0.15, to increase

the size of the training set and reduce overfitting.

ARCH

The VGG16 model consists of 16 layers, including 13 convolutional layers and 3 fully connected

layers. The convolutional layers are grouped into five blocks, with each block consisting of two

or three convolutional layers followed by a max pooling layer. The fully connected layers serve

as a classifier and map the feature vectors extracted by the convolutional layers to the output

classes. We used the Keras framework to implement the VGG16 model with pretrained weights.

Specifically, we used the VGG16 model with the ImageNet weights, which were pretrained on

a large-scale image recognition dataset with 1,000 object categories.

Transfer Learning

To adapt the VGG16 model to our task on the dataset, we used transfer learning. We froze

the weights of all layers in the VGG16 model except for the last three fully connected layers.

We then added a new output layer with 13 nodes corresponding to the 13 object categories in

our dataset. We initialized the weights of the new output layer randomly and trained only the

weights of the output layer while keeping the weights of the rest of the model fixed.

Before applying transfer learning we had almost 21M trainable parameters and after we had only 6M.

Training

We used the Adam optimizer with a learning rate of 0.001 to train the VGG16 model on our

dataset. We used the categorical cross-entropy loss function and monitored the accuracy metric

during training. To prevent overfitting, we used early stopping with a patience of 20 epochs and

a minimum improvement in validation accuracy of 0. We trained the model for 100 epochs and

used a batch size of 32.

Now we move on to the next model which is resnet50 , for data preprocessing we did the same settings as vgg16 but we changed the pixel size to 180x180

Arch

ResNet50 is a deep neural network architecture consisting of 50 layers. The architecture is

built upon the residual block, which allows for the efficient training of very deep networks.

The residual block consists of two convolutional layers and a shortcut connection that bypasses

the convolutional layers. The shortcut connection enables the gradients to flow more easily

through the network, making it possible to train much deeper architectures. In addition to the

residual blocks, ResNet50 includes downsampling layers that reduce the spatial dimension of

the feature maps and increase the number of filters. These downsampling layers are responsible

for extracting more abstract features from the input data. ResNet50 also includes global average

pooling and a fully connected layer at the end of the network, which are used for classification.

The ResNet50 model is loaded using the Keras library and its weights are initialized to pre-

trained weights on the ImageNet dataset. The model is then set up to exclude the top layer,

which is the fully connected layer that performs the classification on the ImageNet dataset. The

input shape is set to 180x180x3, which is the size of the input image, and the pooling is set to

global average pooling. The include top parameter is set to False to exclude the top layer of

the pre-trained ResNet50 model. The classes parameter is set to 13 to indicate the number of

output classes for the new classification task. All the layers in the pre-trained ResNet50 model

are frozen by setting trainable to False. The ResNet50 model is then added to the Sequential

model along with a Flatten layer to convert the output of the ResNet50 model to a 1D array. Two

dense layers are added on top of the ResNet50 model to perform the classification task. The

first dense layer has 512 units with the ReLU activation function, and the second dense layer

has 13 units with the softmax activation function, which is used for multi-class classification.

Transfer Learning

After applying transfer learning we are left with only 1M trainable parameters and 23M non-trainable params.

Training

The code starts by compiling the ResNet50 model with the Adam optimizer with a learning rate

of 0.001, using categorical cross-entropy as the loss function and accuracy as the evaluation

metric. The fit generator function is then used to train the model on the specified number of

epochs (in this case, 100) using the training and validation data generated by the train generator

and test generator, respectively. The verbose parameter is set to 1, which means that progress

bars will be displayed during training. Additionally, the EarlyStopping callback is used to stop

the training process if the validation accuracy does not improve for 20 consecutive epochs.

INCEPTIONV3

Datapreprocessing

To train the InceptionV3 model, we used the ImageDataGenerator class in Keras to perform

data preprocessing and augmentation. The train datagen object was defined with several image

transformation parameters, including a rescaling factor of 1/255 to normalize the pixel values,

a shear range of 0.4, a zoom range of 0.4, horizontal and vertical flipping, and a validation split

of 0.2 to create a separate validation set. These transformations were applied to the training

set images during training to increase the diversity and quantity of the training data, and to

prevent overfitting of the model. Additionally, the validation split allowed us to monitor the

performance of the model on unseen data during training, which helped us to adjust the model

hyperparameters and prevent overfitting.

ARCH

The architecture is built upon the idea of "inception modules", which are convolutional layers

with different filter sizes that are combined to capture features at different scales. InceptionV3

also includes down-sampling layers that reduce the spatial dimension of the feature maps and

increase the number of filters, and global average pooling is used for feature aggregation. The

InceptionV3 model is loaded using the Keras library with pre-trained weights on the ImageNet

dataset. The top layer of the pre-trained InceptionV3 model is excluded by setting include top

to False, and the input shape is set to 224x224x3, which is the size of the input image. All layers

in the pre-trained InceptionV3 model are frozen by setting trainable to False. The InceptionV3

model is then added to a new model using functional API along with a GlobalAveragePooling2D

layer, a dense layer with 1024 units and ReLU activation, a dropout layer with 0.5 rate, and a

dense layer with 13 units and softmax activation, which is used for multi-class classification.

After applying transfer learning we have around 2M trainable params and 21.8M Non-trainable params

Training

In order to train the InceptionV3 model for the classification task, we need to compile the model

with an optimizer, loss function, and evaluation metric. In this case, we use the Adam optimizer

with a learning rate of 0.0001, categorical cross-entropy loss function, and categorical accuracy

as the evaluation metric. To prevent overfitting and improve the generalization of the model,

we use early stopping and learning rate reduction callbacks during the training process. The

early stopping callback monitors the validation loss and stops the training process if there is no

improvement for 5 consecutive epochs, while the reduce learning rate on plateau callback re-

duces the learning rate by a factor of 0.1 if the validation loss does not improve for 2 consecutive

epochs. The model is trained for 25 epochs with a batch size of 32. The fit method is called on

the InceptionV3 model with the training and validation data generators, the early stopping and

reduce learning rate callbacks, and the steps per epoch and validation steps specified to ensure

that all the samples are processed during each epoch.

CNN

Datapre

To train the CNN model, we used the ImageDataGenerator class in Keras to perform data pre-

processing and augmentation. The train datagen object was defined with several image transformation parameters, including a rescaling factor of 1/255 to normalize the pixel values, a shear

range of 0.4, a zoom range of 0.4, horizontal and vertical flipping, and a validation split of 0.2

to create a separate validation set. These transformations were applied to the training set images

during training to increase the diversity and quantity of the training data, and to prevent over-

fitting of the model. Additionally, the validation split allowed us to monitor the performance of

the model on unseen data during training, which helped us to adjust the model hyperparameters

and prevent overfitting.

ARCH

This architecture defines a Convolutional Neural Network (CNN) for image classification. The

model is built using the Keras Sequential API. The input layer is a convolutional layer with 32

filters, a 3x3 kernel, and ReLU activation. This is followed by a max pooling layer with a 2x2

pool size. The model then adds another convolutional layer with 64 filters and a 3x3 kernel,

and another max pooling layer. The process is repeated with two more convolutional layers,

one with 128 filters and the other with 256 filters, each followed by a max pooling layer. The

output from the convolutional layers is flattened and passed to a dense layer with 512 neurons

and ReLU activation. A dropout layer is added to prevent overfitting. Finally, the output is

passed to a final dense layer with 13 neurons and softmax activation for classification into 13

classes. The model is compiled using categorical crossentropy loss and the Adam optimizer.

WE have 19M Trainable params

Training

The code initializes two callbacks, EarlyStopping and ReduceLROnPlateau, which are used to

prevent overfitting and optimize the learning rate, respectively. The batch size is set to 32 and

the model is trained for 50 epochs on the training dataset using the fit() method. The validation

dataset is also passed in as a parameter, allowing the model to be evaluated on data it has not

seen before. The steps per epoch and validation steps are set based on the number of samples

in the respective generators. Finally, the training progress is stored in the history variable for

later analysis. The early stopping callback monitors the validation loss, stopping the training

process if there is no improvement after 5 consecutive epochs. The reduce learning rate on

plateau callback is used to reduce the learning rate by a factor of 0.1 if there is no improvement

in the validation loss after 2 epochs. This allows the model to make smaller adjustments to its

weights and avoid overshooting the optimal solution.

RESULTS

VGG-16

The model was trained for 100 epochs, and the training and validation accuracy and loss were

recorded after each epoch. The model achieved an accuracy of 89.1% on the training data and

82.3% on the validation data after the first epoch. The accuracy on the training data continued

to increase, reaching a maximum of 96.4% after the 14th epoch. The validation accuracy also

increased over time, reaching a maximum of 90.8% after the 16th epoch.

The loss on the training data started at 0.7615 and decreased over time, reaching a mini-

mum of 0.1574 after the 17th epoch. The loss on the validation data also started high at 1.4022

but decreased over time, reaching a minimum of 0.7741 after the 10th epoch. The high accu-

racy achieved on the training data suggests that the model has learned to classify the sketches

well, and the increasing validation accuracy suggests that it is generalizing well to unseen data.

However, it is worth noting that the validation accuracy never exceeds the training accuracy

by a significant margin, which may suggest that the model is slightly overfitting to the training

data. Nevertheless, the model’s performance is still very good, and it may be worth further

fine-tuning or tweaking the model to optimize its performance on this specific dataset.

Resnet-50

The model achieved an accuracy of 83% on the training set and 69% on the validation set in the first epoch. Over the course of the next 20 epochs, the model continued to improve, achieving an accuracy of 96.7% on the training set and 83.85% on the validation set by the 20th epoch. It is noteworthy that the model’s performance on the validation set initially lagged behind its performance on the training set, but this trend eventually reversed, and the model consistently outperformed the training set by the 20th epoch. This suggests that the model may have been overfitting in the early epochs, but eventually learned to generalize to new data. It is also interesting to note that the validation accuracy showed a somewhat fluctuating trend over the course of the 20 epochs, with occasional dips and spikes. This is not unusual in deep learning models, and may be attributed to the stochastic nature of the optimization process.

InceptionV3

The results from the training show that the model is improving over time, with both loss and categorical accuracy improving as the epochs progress. In the first epoch, the model had a categorical accuracy of 25.84% and a validation categorical accuracy of 50%. However, the accuracy improved significantly over the next few epochs. By the fifth epoch, the model had a categorical accuracy of 71.96% and a validation categorical accuracy of 82.81%. In the following epochs, the validation categorical accuracy remained relatively stable at around 82-86%, while the categorical accuracy continued to improve, peaking at 84.97% in the twelfth epoch. The initial loss of 2.3508 and categorical accuracy of 0.2584 on the first epoch improved significantly on the last epoch with a loss of 0.3898 and categorical accuracy of 0.9070. Figure 5.29: Loss and Accuracy Graphs of Inception-V3 Model while training on the dataset. Looking at the validation results, the validation loss and validation categorical accuracy are both improving over time, indicating that the model is generalizing well to the validation data. The validation categorical accuracy starts at 50% on the first epoch and improves to 90.70% on the last epoch. Similarly, the validation loss starts at 1.6705 on the first epoch and decreases to 0.3898 on the last epoch. Based on the trend, it seems that the model has not yet converged, and training for more epochs could yield further improvements in both the training and validation results. It is important to note that the model’s performance may plateau after a certain number of epochs, so monitoring the validation results and avoiding overfitting is essential. One interesting trend that we can observe is that the model’s categorical accuracy on the validation set is consistently higher than its categorical accuracy on the training set. This phenomenon is known as generalization, and it indicates that the model is learning to recognize patterns and features that are relevant to the task at hand, rather than simply memorizing the training data.

CNN

The CNN model’s learning process on the given dataset showed a gradual improvement in accuracy and loss over the 50 epochs. In the first epoch, the model had a low accuracy rate of 0.0726 and a high loss value of 2.7748, while the validation accuracy was slightly higher at 0.0859 and the validation loss was 2.5575. However, as the training progressed, the accuracy improved, and the loss decreased. By epoch 13, the model had achieved an accuracy of 0.6723 and a loss of 0.9800, indicating that the model was learning well. At epoch 15, the learning rate decreased by a factor of 10, and there was a slight decrease in validation accuracy. This decrease may have been due to overfitting, as the model was adjusting to the new learning rate. However, the model continued to train, and by epoch 50, the accuracy had significantly improved from 0.0726 to 0.7010, while the loss had decreased from 2.7748 to 0.9205. Overall, the results indicate that the CNN model was able to learn and generalize well on the given dataset, achieving high accuracy rates and demonstrating good performance in terms of loss. The trend of gradual improvement in accuracy and loss over time is a positive indication of the model’s ability to learn from the data and make accurate predictions.

After we saw the results of each model , its time to test each model to see if it can recognize my drawing style.First off we have a demonstration for a model trying to recognize my drawings then we will see some testing results for all the models .

VGG-16

The VGG-16 algorithm was highly accurate in classifying airplane, apple, and banana sketches due to their simple nature, but encountered difficulty with bicycle sketches, likely due to the complexity of the image. The algorithm also struggled with moon sketches, sometimes misclassifying them as bananas due to their similar shape. The model had mixed results with sheep and dog sketches due to the diversity in the training sketches. The model did not face any difficulties with door, ladder, table, and tree sketches due to their simplicity. VGG16 had a peculiar challenge with classifying wheel sketches, even after adding details, suggesting that a larger dataset may be needed to improve accuracy

RESNET-50

ResNet50 struggles with accurately classifying complex sketches, including airplanes, bananas, cars, bicycles, and dogs, even after adding more details to the sketches. ResNet50 performs well in recognizing simple sketches like apples, tables, wheels, and doors. It occasionally misclassifies tables as ladders and wheels as moons or apples when only a simple circle is drawn without inner details. Overall, ResNet50’s performance on complex sketches is challenging, and further improvements in the training process and model architecture may be necessary to address this issue.

INCEPTIONV3

The model struggled with classifying airplane sketches, apples, and bicycles due to similarities in shape or complexity of the drawings. However, adding more details to the sketches improved the accuracy of classification. Inceptionv3 did not have difficulty with classifying simple sketches such as bananas, doors, ladders, tables, and wheels. Moon sketches were challenging for Inceptionv3, similar to VGG16, due to their resemblance to the shape of a banana. Sufficient and specific details were required for Inceptionv3 to accurately classify dog sketches

CNN

The CNN encountered significant difficulties in classifying airplane sketches due to their diversity and complexity. However, adding additional details to the sketches such as landing gear, windows, and tail improved the accuracy of classification. The CNN had no difficulties in classifying apple, bicycle, door, and ladder sketches. The CNN struggled to classify car sketches and failed to classify banana, dog, moon, sheep, table, and wheel sketches. The shape and complexity of the objects, as well as similarity to other objects, were identified as factors that affected the CNN’s ability to classify the sketches accurately.

Finally we have this graph that represents the accuracy and validation accuracy of all the models compared together , and if we zoom in we can see that overall transfer learning performed much better than traditional machine learning in sketch recognition on our dataset.