UNIVERSITY OF CALIFORNIA
Los Angeles

Augmenting MRI Classification Models with Synthetic Images Created via Generative Models

A dissertation submitted in partial satisfaction
of the requirements for the degree
Doctor of Philosophy in Statistics

by

Daniel Kwon

2024

ABSTRACT OF THE DISSERTATION

Augmenting MRI Classification Models with Synthetic Images Created via Generative Models

by

Daniel Kwon
Doctor of Philosophy in Statistics
University of California, Los Angeles, 2024
Professor Yingnian Wu, Chair

WIP - Check back later

The dissertation of Daniel Kwon is approved.

...

...

Yingnian Wu, Committee Chair

University of California, Los Angeles

2024

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

Introduction

With the performance of state-of-the-art generative models improving rapidly, synthetic images can be produced with ease using simple text or image prompts. Image generation models such as Dall-E or Stable Diffusion are able to create images with sufficient quality that the margin of difference between real and synthetic images is becoming increasingly thin. As the level of effort in generating synthetic images decreases and the quality of these images increases, the potential to use synthetic images as a means to augment image datasets becomes increasingly viable—especially in fields where gathering images is constrained by costs or other resources.

In the field of medical imaging, where image datasets often require specialized equipment and subject matter experts to capture and label images, gathering enough data to sufficiently train a classification model can be both expensive and time-consuming. For example, with the cost of magnetic resonance imaging (MRI) ranging from $1,600 to $8,400 in the United States, a dataset consisting of a few hundred images can exceed $1 million; image classification models often require thousands of photos.

One way computer vision models have historically attempted to remedy insufficient training datasets has been to employ traditional image augmentation techniques, which involve applying transformations such as flipping or blurring an original image to produce additional images for training. However, such transformations must be applied carefully in order to not lose the fidelity needed to make accurate diagnostic predictions. Transformations that alter the nature of the images can potentially lead to a degradation in model performance.

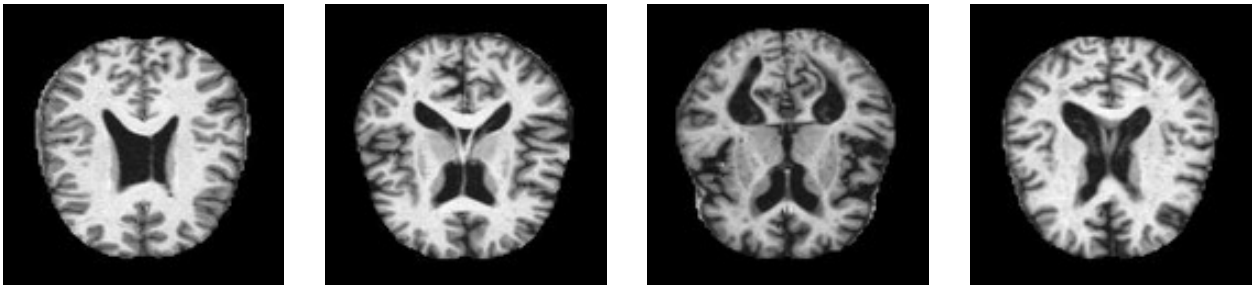
Augmenting image datasets with high quality synthetic images may allow for significant cost-saving while maintaining model performance, and previous research has found that synthetic images may play a complimentary role to image augmentation. The goal of this paper is to train image classification models on a MRI dataset to identify the presence of varying levels of dementia and investigate the effect of different image augmentation policies on model performance as well as compare their performance against an augmentation policy that generates synthetic images.

CHAPTER 2

Dataset

2.1 Alzheimer MRI Preprocessed Dataset

For the purposes of this paper, we use publicly available MRI images of patients with varying levels of dementia, labeled as *not demented*, *very mildly demented*, *mildly demented*, and *moderately demented*.



(a) Not Demented (b) Very Mildly Demented (c) Mildly Demented (d) Moderately Demented

Figure 2.1: Examples of real MRI images

Not Demented	Very Mildly Demented	Mildly Demented	Moderately Demented
2566	1781	724	49

Table 2.1: Count of each class in training data

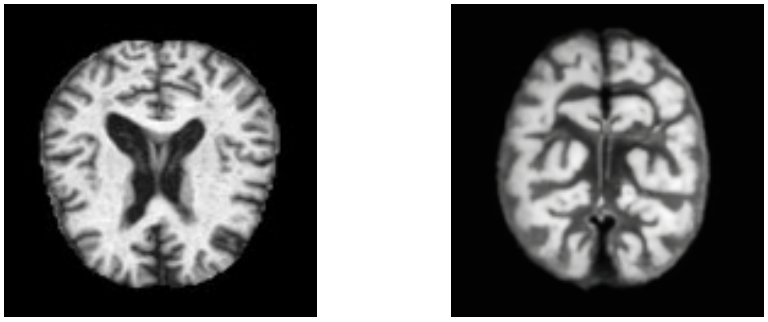
These images were downloaded via the datasets module provided and maintained by

Hugging Face. All images are in black and white and have been pre-processed to 128x128 resolution or 256x256 resolution, depending on the model being tested.

2.2 Synthetic Images

This paper utilizes OpenAI's Dall-E-2 to produce synthetic images. To generate synthetic images, an original image is supplied as a source and the generative models are prompted to creating a similar image.

In order to generate images using Dall-E-2 we utilize OpenAI's image variation API endpoint, which returns a variation of a given image.



(a) Real Image (b) Synthetic Image

Figure 2.2: Results of Dall-E-2's image variation generation

Dall-E-2 reliably generates images that are similar to the real MRIs that are provided. Given that dementia is often physically tied to brain atrophy in certain areas, the synthetic images produced by Dall-E-2 generally reproduce the ridges, folds, and cavities of the source images as well.

CHAPTER 3

Exploration using Grad-CAM

In order to better understand what parts of an image classification models deem to be more "important" in regards to predicting the presence of dementia, we employ Gradient-weighted Class Activation Mapping, or Grad-CAM, as a way to represent what a convolutional neural network "sees" when classifying both real and synthetic images. Grad-CAM is a technique that maps the gradients of a final convolutional layer in regards to a specific class prediction to product a heat map. The result is a visually intuitive way of illustrating which portions of an image contribute most to a given classification [source here].

3.1 Overview of Grad-CAM

1. First, calculate the gradients of the model's output in the final convolutional layer, with respect to the feature maps. Assuming y_c is the score for class c (before softmax) and δA^k is the feature map activation of the kth layer, compute $\frac{\delta y_c}{\delta A^k}$.

2. Next, calculate the global average pooling of the feature map. Global average pooling is a pooling operation designed to generate one feature map for each corresponding category of the classification task in the last [convolutional] layer and then take the average of each feature map [source here].

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\delta y_c}{\delta A_{ij}^k}$$

Z here represents spatial dimensions of the feature map—in this case, its height and

width. By dividing by Z, we obtain the average of gradients over all spatial locations. $\frac{\delta y_c}{\delta A_{ij}^k}$ is the gradient for class c with respect to the activation at A_{ij}^k, or spatial location (i,j) on activation layer k.

3. Lastly, we take the resulting importance weights in a_k^c and combine them with A^k to calculate a weighted combination of all forward activation maps. We then apply the ReLU activation function to the resulting combination in order to take the positive values only. This ensures that we are only looking parts of the image that are positively correlated with a given class.

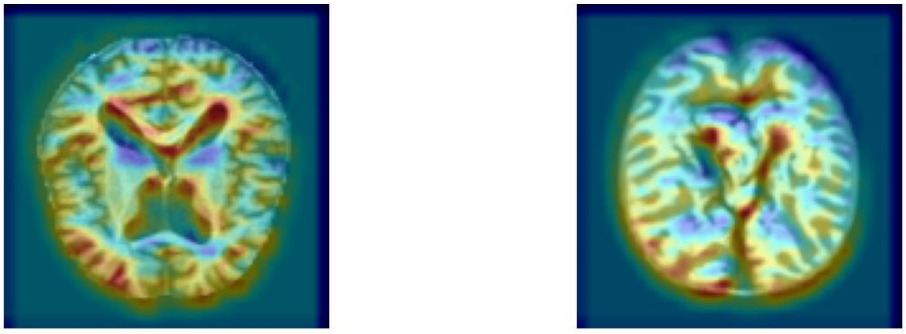
$$L_{Grad-CAM}^c = ReLU(\sum_k \alpha_k^c A^k)$$

Notice that $L_{Grad-CAM}^c$ will have the dimensions of the final convolutional layer and therefore is likely to be smaller than the original input image. In that case, we simply upsample the result to the dimensions of our original image in order to overlay the two.

3.2 Exploratory Results

In order to better understand what our models "look at" when evaluating an image and predicting a class, we take a basic CNN with the same architecture outlined in chapter 5, section 1 and train it on real MRI images. While the model architecture is relatively simple compared to deeper networks, it is sufficient in order to explore what parts of an image a typical CNN focuses on. For the purposes of this exploratory analysis we only look at MRIs labeled as moderately demented or non-demented, as these classes are the at the most extreme ends of the spectrum and will illustrate what characteristics contribute to the model detecting the presence of absence dementia.

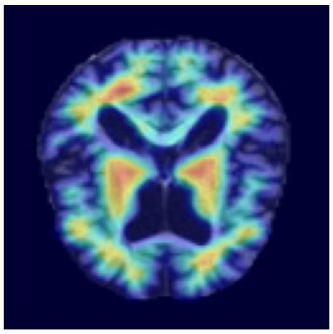
3.2.1 Real vs synthetic images labeled as having moderate dementia



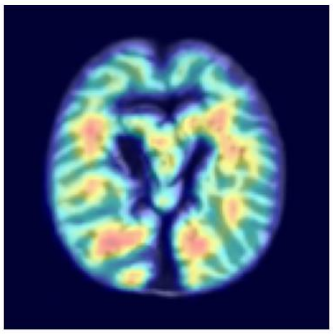
(a) Real (b) Synthetic

Figure 3.1: Grad-CAM, Predicting: ModerateDemented

When looking at a real image of a brain with moderate dementia, we can see that signs of atrophy contribute to the likelihood that the model classifies this brain as one with moderate dementia. When applying Grad-CAM on a synthetic image generated via the previous original image as a source, we can see that the qualities that indicate the presence of dementia are also present in the synthetic images as well. While not as pronounced, the areas of atrophy in the synthetic brain image are present and our model focuses on those areas in both real and synthetic images.



(a) Real

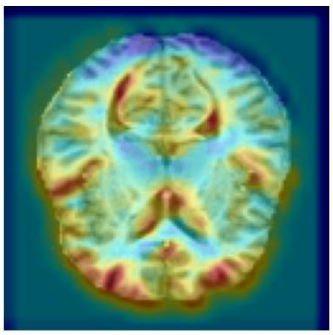


(b) Synthetic

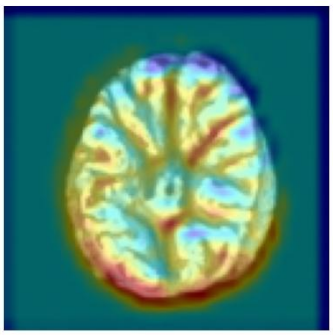
Figure 3.2: Grad-CAM, Predicting: ModerateDemented

The same image, shows that the areas of non-atrophy, particularly those closer to the brain stem, contribute to a likelihood that this image would be classified as non-demented. The associated synthetic image shows similar patterns.

3.2.2 Real vs synthetic images labeled as having no dementia



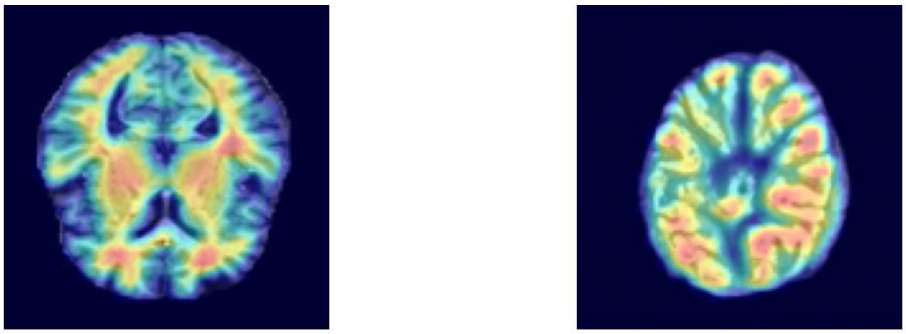
(a) Real



(b) Synthetic

Figure 3.3: Grad-CAM, Predicting: ModerateDemented

When comparing real and synthetic images of a brain with no signs of dementia, we see that our synthetic image captures the areas of non-atrophy that is present in the original source image.



(a) Real (b) Synthetic

Figure 3.4: Grad-CAM, Predicting: ModerateDemented

There does not appear to be as signficant of atrophy present in an image of a non-demented brain—which our synthetic image has adequately captured. Both real and synthetic images show larger non-atrophied areas of the brain, which our model is focusing on as an indication of a brain without dementia.

CHAPTER 4

Background on Training Deep Learning Models

4.1 Overview of Neural Networks

While deep learning architectures can span many layers and can incorporate many different mechanisms, at its core all deep learning models are neural networks. Training neural networks comprise of a few key steps.

First, training data is input into a neural network and passed through as-is in what is known as a "forward pass." In a fully connected neural network, every individual neuron in a layer of the neural network is connected to each neuron in the preceding layer

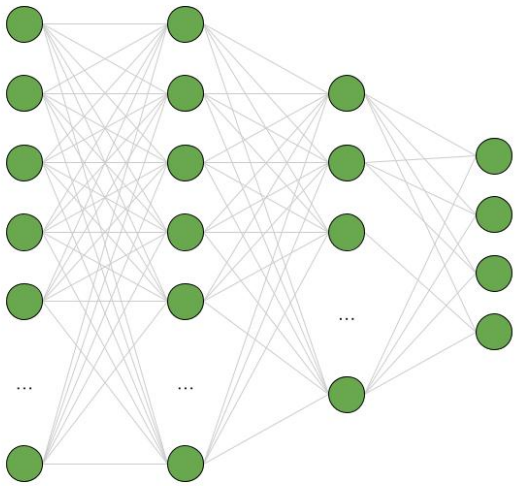


Figure 4.1: An illustration of a typical fully-connected neural network

Each of these connections has a weight that stores how strong of a connection each

preceding node has to the current node. A bias term is also present, which represents whether a neuron is general activated or not. Often, an initialized neural network will comprise of randomized weights and biases whereas a pre-trained model will have the weights and biases already defined from previous training. When training a neural network, the weights and biases are what are being adjusted in order to minimize loss. The formulation of a single neuron with n nodes in the previous layer is below:

$$\sigma(w_0a_0 + w_0a_0 + w_0a_0 + ... + +w_{n-1}a_{n-1} + b)$$

After the forward pass, backpropagation occurs in which the gradient (the vector of partial derivatives) of the loss function with respect to each weight and bias is calculated via chain rule. Each gradient value represents the magnitude and direction of the change in our loss function given a change to that particular weight or bias. Because backpropagation uses the chain rule to propagate the loss backward from the output layer to the input layer, the process of finding the gradient of a loss function remains the same no matter how many layers are in the network or how many neurons are in each layer.

Once the gradient is calculated, the gradient descent algorithm is used to update the weights and biases in order to minimize loss. However, applying the gradient descent on an entire training dataset in a single batch is computationally expensive. Instead, gradient descent is often applied to subsets of the training data, called mini-batches. Each training iteration, or epoch, will then take a mini-batch to apply the forward and backward passes on to to updates its weights and biases. The result is an accurate approximation of the gradient of the loss function while significantly decreasing computational expenses [source].

4.2 Loss Function

The loss function used throughout this paper when measuring model performance is cross entropy loss. Cross entropy loss is defined as:

$$L = -\sum_{c=1}^{C} y_c log(p_c)$$

where C = the number of classes, y_c is the true label for class, and p_c is the predicted probability for class c. Cross entropy loss therefore compares the predicted probabilities with the actual labels and penalizes more when the predicted probabilities for a correct class is low [source].

Cross entropy loss is better suited for measuring the performance of classification models when multiple classes are involved than traditional measures of error, such as the sum of residuals, as it is more sensitive to predictions that are "more" incorrect. In an image classification problem, a model may mislabel a given image, but the cross entropy loss will be lower if the predicted probability for the correct class is higher, even if the model did not ultimately end up labeling correctly. Contrast that with the sum of residuals, which only views predictions as completely correct or completely incorrect and fails distinguish between the degrees of how right or wrong a prediction can be [source].

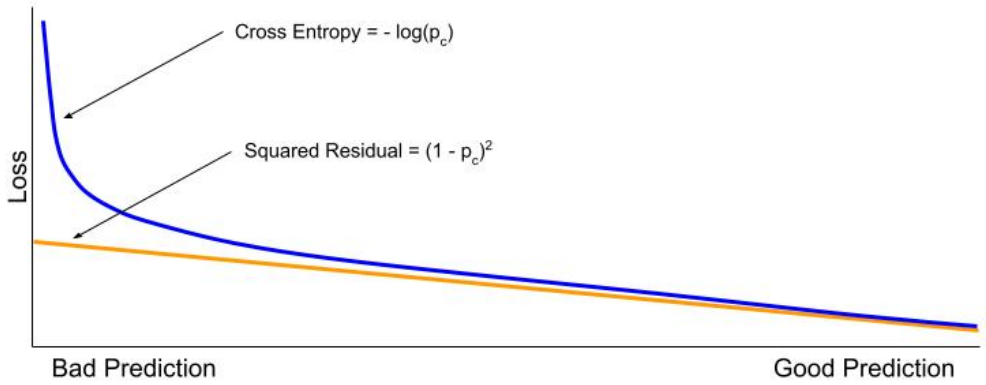


Figure 4.2: Illustration of Cross Entropy Loss vs Squared Residual

Take for example, when the true label is 1 and the predicted class weight y_c is 0.0001—

making our model prediction very bad. The squared residual would be $(1-0.0001)^2 = 0.9998$ while our cross entropy loss would be more punitive at $-log(0.0001) = 4$. Cross entropy loss also has a much larger gradient for very bad predictions, allowing our model to more quickly learn to avoid bad predictions.

4.3 Gradient Descent

As mentioned previously, performing gradient descent on every single training image can be computationally inefficient. Instead, stochastic gradient descent which divides the training dataset into smaller batches and applied the gradient descent algorithm on each batch, can significantly reduce runtime.

In addition to dividing the training dataset into mini-batches, there are other methods that make optimization via gradient descent more efficient—namely, learning rate and momentum.

4.3.1 Learning Rate

The formulation for stochastic gradient descent is defined as:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t; x^{(i)})$$

Where:

- θ_t is the model's weights and biases at iteration t
- η is the learning rate
- $\nabla_\theta L(\theta_t; X^{(i)})$ is the gradient of the loss function at time t with respect to $x^{(i)}$
- $x^{(i)}$ is a randomly selected mini-batch

The learning rate is a critical parameter for the training of any deep learning model and can dramatically effect how the gradient descent algorithm behaves. Learning rate can be understood intuitively as the step size the optimization algorithm takes when it decides which direction to step—too large of a learning rate can lead to sub-optimal changes to a network's weights and biases; too small and achieving model convergence can much longer.

Furthermore, the learning rate does not necessarily need to be constant throughout the entire optimization process. It can vary over time with a larger learning rate at the beginning of the training, when large steps along the gradient is appropriate, and reducing later in training process, when smaller learning rates are more appropriate.

4.3.2 Momentum

Momentum takes into account the gradient at previous steps, beyond simply looking at the gradient at time $t-1$. By doing so, the gradient descent algorithm tends to stay in the direction where the gradient has consistently pointed towards. The formulation for stochastic gradient descent with momentum is defines as:

$$\theta_{t+1} = \theta_t - \eta v_t$$

Where v_t is known as the velocity term, defined as:

$$v_t = \beta v_{t-1} + (1-\beta)\nabla_\theta L(\theta_t)$$

Where:

- β is the momentum coefficient, which determines how much of the previous gradients carry over into the latest velocity calculation at time t—this parameter can be set between 0 and 1
- $\nabla_\theta L(\theta_t)$ is the gradient of the loss function at time $t-1$

4.3.3 Adam Optimizer

Adaptive Moment Estimation, also known as Adam Optimization, combines aspects of both a variable learning rate as well as momentum in order to develop an optimization algorithm that efficiently calculates the gradient. Gradient descent using Adam optimization is defined as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

Where:

- $\hat{m}_t$ is known as the bias correction for the first moment and defined as $\frac{m_t}{1-\beta_1^t}$
- $\hat{v}_t$ is known as the bias correction for the second momemt and defined as $\frac{v_t}{1-\beta_2^t}$
- $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta L(\theta_t)$
- $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta L(\theta_t))^2$
- β_1 is the decay rate for the first moment
- β_2 is the decay rate for the second moment

The first moment m_t allows the Adam optimization algorithms to take into account momentum by taking the weighted average of past gradients. The second moment v_t acts as an adjustment to the learning rate and allows Adam algorithm to have a variable learning rate based on previous squared gradients. For all models trained in this paper, we use the Adam optimizer.

4.4 Fine-tuning Pre-trained Models

Often times, computer vision models require a large number of images as well as a significant amount of computation time. In order to accelerate the training process, many image classification models will utilize a pre-trained model, instead of starting training from randomly

initialized weights and biases. The most common image dataset used to pre-train models is the ImageNet-1000 dataset, which is comprised of 1000 distinct classes, 1,281,167 training images, 50,000 validation images and 100,000 test images.

Once a model is pre-trained, fine-tuning can take two forms—either progressing through training as you normally would with images intended for fine-tuning (i.e., your dataset) or freezing all weights in the pre-trained model (essentially locking the model from any further training) and re-training only the fully connected neural nerwork appended to the pre-trained model. If the latter method is used, the pre-trained model can regarded as a fixed feature extractor for an image. For the purposes of our paper, all pre-trained models are fine-tuned by re-training on our dataset.

CHAPTER 5

Overview of Model Architectures

In order to gain an understanding as to how both convolution-based and attention-based architectures react to the introduction to synthetic data in training, We tested three different deep learning model architectures for our image classification models—a basic Convolutional Neural Network (CNN), a pre-trained ResNet-18 model, and a pre-trained Vision Transformer (ViT).

5.1 Convolutional Neural Networks

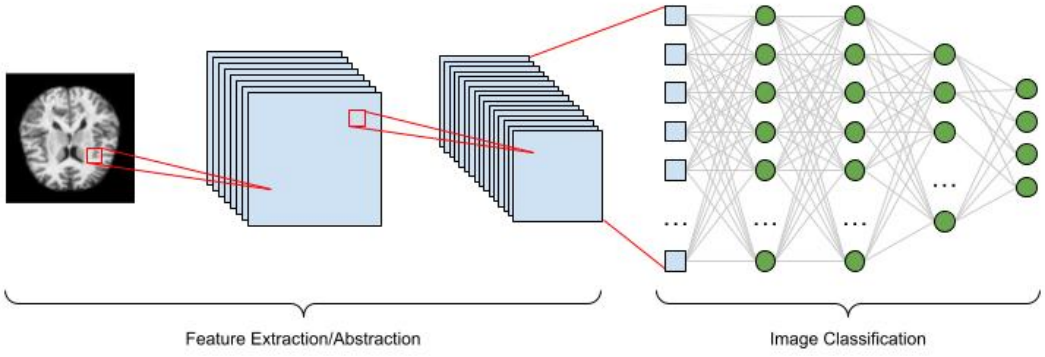


Figure 5.1: Typical CNN Architecture

CNNs are a deep learning architecture that is comprised of convolutional layers—which abstract an image to a feature map, pooling layers—which reduce the dimensions of data by combining the outputs of adjacent layers via downsampling, and fully connected layers—

which are neural networks that take the final activations from the convolutional and pooling layers to generate class weights.

Convolutional layers are a key component of convolutional neural networks and act as a way for the network to extract abstract features from an image. The primary operating mechanism of a convolutional layer comprises of a kernel, or a filter that is applied to the original image.

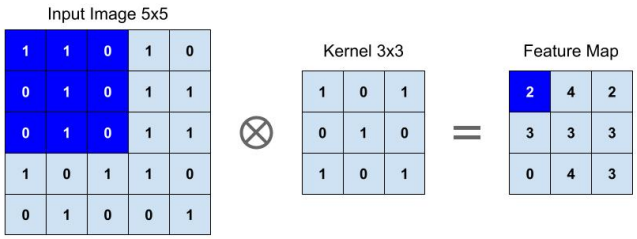


Figure 5.2: Illustration of Convolutional Layer

Often, convolutional layers may stack multiple kernels resulting in activation maps with more channels. As subsequent layers get deeper, more complex and increasingly abstract features are extracted from the original image. While the kernels from the earlier layers of a CNN may learn to identify basic shapes and patterns, kernels in deeper layers may learn to identify specific edges and details.

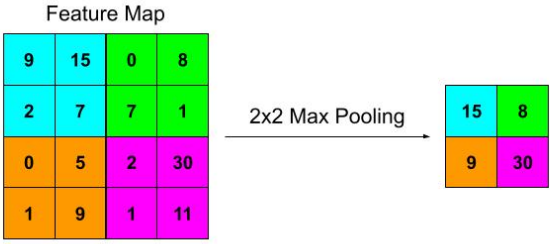


Figure 5.3: Illustration of 2x2 Max Pooling

Though not explicitly required, max pooling layers frequently follow convolutional layers. Max pooling layers essentially downsample the activation map so that the important features and spatial relationships contained in an image are captured while removing unnecessary details and reducing the feature map's dimensions, thereby improving computational efficiency of the network.

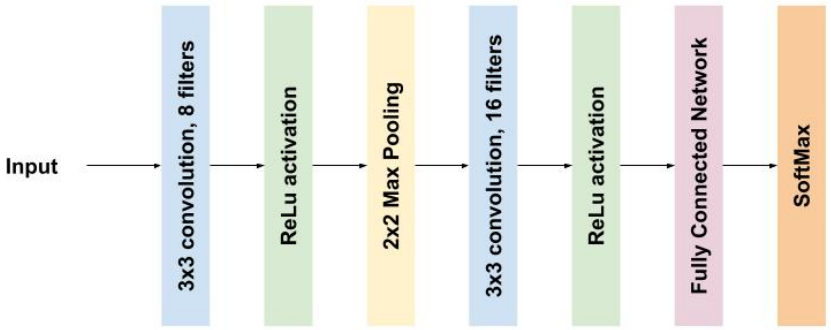


Figure 5.4: Basic CNN Architecture

For the purposes of this paper, the CNN we train from scratch consists of 2 convolutional layers, each paired with a ReLU activation function and a max pooling layer. After all feature extraction layers are complete, the final activation layer is fed into a fully connected neural network with 2 hidden layers and an output layer that predicts probabilities weights for each of our four classes.

5.2 ResNet-18

The second model architecture we test is a ResNet-18 model, pre-trained on the ImageNet-1000 dataset. The ResNet architecture is similar to a basic convolutional neural network but adds two additional components—skip connections and residual blocks.

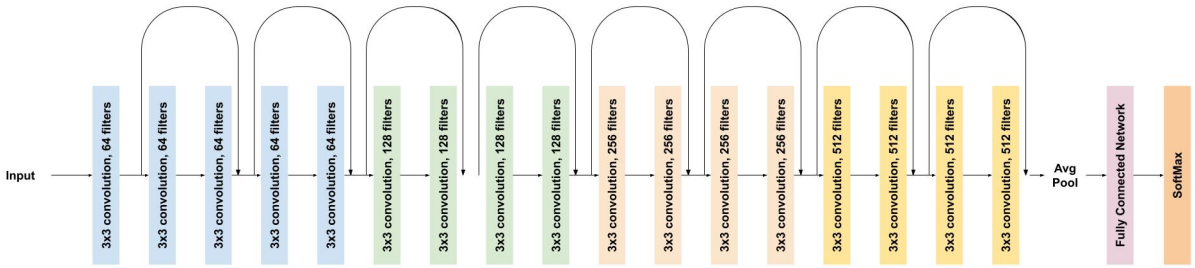


Figure 5.5: ResNet-18 Architecture

Skip connections allow the network to skip a convolutional layer entirely thereby skipping training for some layers. Doing so helps to avoid a phenomenon known as vanishing gradients—which occurs in deep networks where the gradient calculated via backpropagation becomes so small that weights from earlier layers become insignificant and changes to the input image cease to meaningfully affect feature mapping in deeper layers. Skip connections allow the ResNet model to build deep networks while avoiding vanishing gradients by giving the model a mechanism in which training a layer can be skipped entirely.

ResNet architectures are also characterized by their use of residual blocks, which consistent of 2 convolutional layers and a skip connection. Residual blocks allow the ResNet model to train on the residual between layers, or the difference between the input and the ideal output of the layers.

$$y = F(x) + x$$

Where:

- x is input fed into the residual block
- $F(x)$ is output coming from the residual block
- y is desired final output coming from the residual block

Therefore, $F(x) = y - x$ makes $F(x)$ the residual of y and x. ResNets train by using residual blocks to learn the residual mapping, instead of being forced to train each convolutional layer and the desired feature map (although that is still an option left for the model to evaluate). By doing so, the ResNet remains computationally efficient while maintaining the ability to extract features, either through its traditional convolutional layers or via a residual block.

5.3 Vision Transformers

The last type of model architecture tested—known as vision transformers or ViT for short—is one that foregoes the use of convolutional layers entirely and instead utilizes the attention mechanism to capture spatial relationships and abstract features.

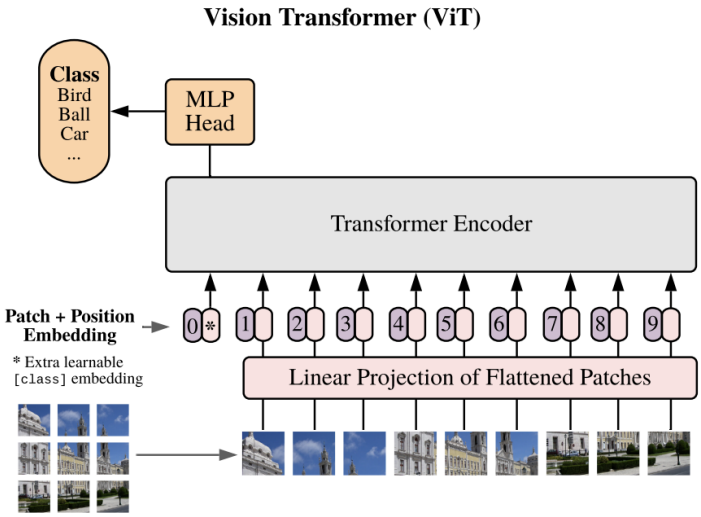


Figure 5.6: ViT Architecture

The transformer architecture is often utilized in natural language processing tasks and serves as the architectural foundation for cutting edge large language models such as ChatGPT, Llama, Gemini, and Claude.

Because applying transformers for the purpose of image classification closely parallels how they are employed for text generation, each transformer component reviewed in this section will be accompanied with an analogous example of how the component would work in a text transformer whenever possible.

5.3.1 Background on Embeddings

Before any data can be used as an input into a transformer, it must first be turned into a numeric representation so that the appropriate operations can be performed on it. For example, if an embedding has three dimensions you can imagine the embeddings for words such as "jump" and "leap" to have high cosine similarity, even if they are different words.

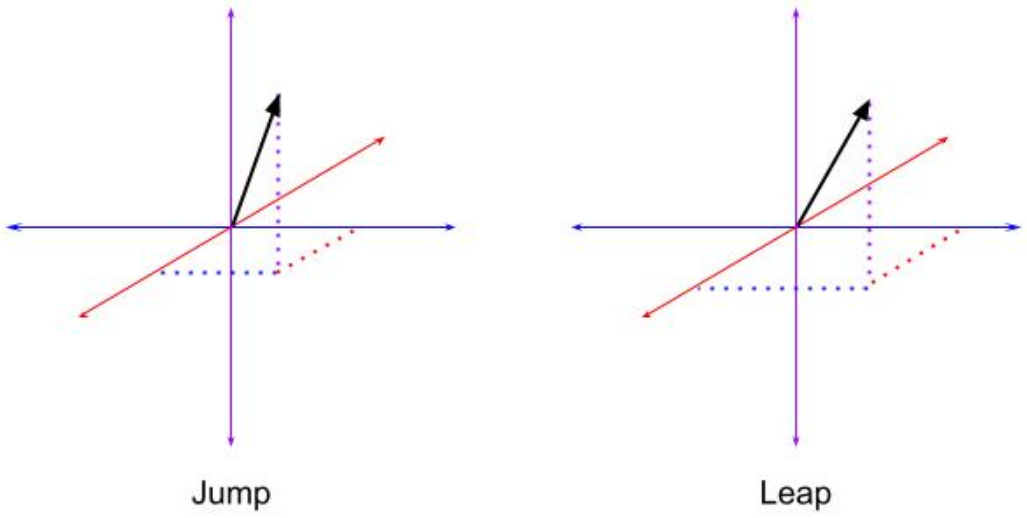


Figure 5.7: Hypothetic 3-dimensional word embeddings

In practice, embeddings can be tens of thousands of dimensions. It is also worth noting that these embeddings are not static representations but trainable parameters that can be modified and adjusted as the transformer trains.

5.3.2 Patch Embeddings and Positional Encoding

In a ViT, rather than words (or tokens) being transformed into embeddings, the image is broken up into a series of 16x16 images and flattened into a 256 dimensional vector. An original image that is $x \in \mathbb{R}^{H \times W \times C}$ is therefore split into patches that are $\mathbb{R}^{P^2 \times C}$. Therefore across N patches, we have matrix $x_p \in \mathbb{R}^{N \times (P^2 \cdot C)}$.

Where:

- H and W is the height and width of the original image
- C is the number of channels in the original image
- P is the dimension of the $P \times P$ patch
- $N = HW/P^2$ is used to calculate the needed number of patches, given a $P \times P$ patch size

x_p is then unrolled to a vector and multiplied by an embedding matrix E to get the patch embedding.

$$z_i = x_p \cdot E$$

Where:

- $E \in \mathbb{R}^{(P^2 \cdot C) \times D}$ is a projection matrix with learnable weights
- x_p is a given image patch, unrolled into vector form

The patch embedding is then combined with the positional encoding by summing it with the patch embedding in order to give the transformer the ability to track where each image patch is relative to the original image. The positional embedding represents the relative position of the original image that the image patch is sourced from. The resulting patch

embedding and position encoding are then fed into a transformer architecture similar to how a string of token embeddings are fed into a traditional transformer used for natural language processing.

5.3.3 Transformer Architecture

The transformer encoder in a ViT consists of alternating layers of multi-headed self attention and MLP blocks, with layer normalization applied before every block and residual blocks after every block.

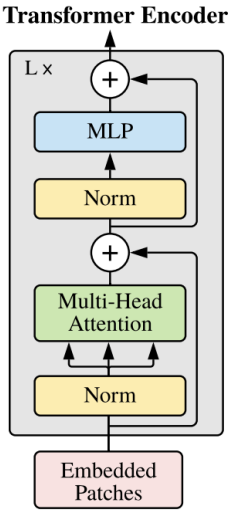


Figure 5.8: ViT Encoder Architecture

The ViT architecture is notable for its usage of the multi-head attention block, which combine query (Q), key (K), and value (V) matrices in order to allow the model to "pay attention" to certain patches.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Where:

- Q, K, and V are the query, key, and value matrices, respectively
- d_k is the dimensions of the key vectors, such that dividing by $\sqrt{d_k}$ ensures that the variance of $q \cdot k$ is 1

In order to pay attention to different image patches under different contexts the ViT architecture uses multi-head attention, which allows for multiple attention mechanisms in parallel.

$$MultiHead(Q, K, V) = [head_1, head_2, ..., head_h]W_0$$

Where:

- $head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$
- W are all learnable parameter matrices

Intuitively, multi-head attention allows the ViT model to pay attention to different parts of the image for different contexts. For example, certain characteristics of an image may warrant more attention when defining segment boundaries, while those same characteristics may not warrant attention when considering segment orientation.

CHAPTER 6

Overview of Image Augmentation Techniques

In addition to different model architectures, we also test combinations of different traditional image augmentation policies and image augmentation via synthetic images. See below for all image augmentation pairings tested [**?**]:

Image Augmentation Policies

- Random horizontal flipping
- AutoAugment policy
- No image augmentation

Synthetic Image Policies

- Synthetic image generation via Dall-E-2
- No synthetic image augmentation

6.1 Rudimentary Image Augmentation

Image augmentation in its simplest basic form is a policy that consists of basic image transformations, such as flipping an image on its horizontal or vertical axis. Often, this image augmentation is applied randomly based on a probability defined prior to training. Because rudimentary image augmentation is cost effective and can result in improved performance for a image classification model with relatively low computational cost, it is frequently used to augment limited datasets are difficult or expensive to collect and/or label.

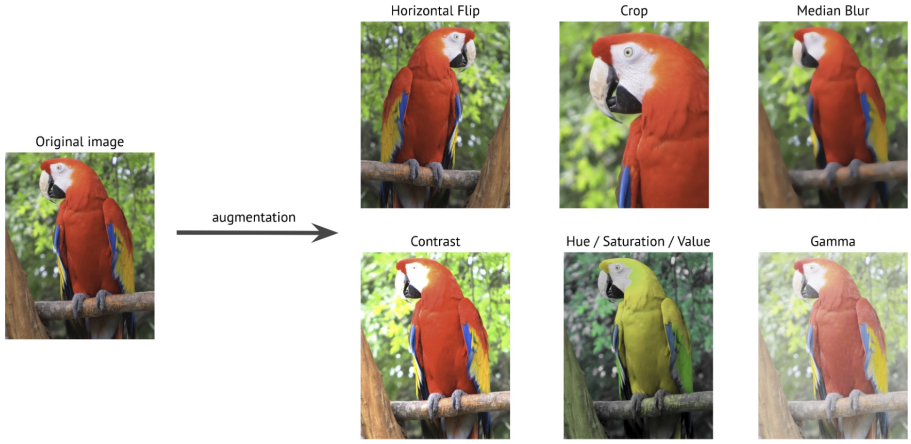


Figure 6.1: basic image augmentation examples

For the purposes of this paper, the basic image augmentation policy applies a horizontal flip randomly with a probability of 50%. Other image transformations, such as cropping or adjusting the color saturation, were not tested due to the nature or our training images. The MRI dataset is in grayscale and intended to capture the entirety of the brain and therefore cropping or changing the color of the image in any way did not fit our particular use case.

6.2 AutoAugment

AutoAugment is an image augmentation policy that was developed by researchers at Google. It takes existing augmentation procedures, such as the ones listed in the previous section, and automates the selection of which procedures to include, in order to select the optimal transformations in order to improve the model performance. The result is a systemic augmentation policy that eliminates the need for manual tuning.

The AutoAugment policy works by employing a reinforcement learning framework in order to find optimal policies. Using a neural network designated as the controller, the AutoAugment policy explores a defined search space of possible transformations—with each transformation having a probability of being applied and a magnitude of the transformation.

The controller then tests augmentation policies within the search space in order to train a model. The model's subsequent performance is measured against a validation dataset and is used to further optimize the image augmentation policy. This feedback loop continues until controller has found an optimal augmentation policy.

AutoAugment is more computationally expensive than a basic image augmentation policy, as it involves training the controller neural network before it can be used in the training of the actual model itself, which can be an entirely separate deep learning model. To address this, AutoAugment policies can be pre-trained on larger datasets—which are then generalized to other images. In this paper, we tested an AutoAugment policy pre-trained on the popular and widely available IMAGENET dataset.

6.3 Synthetic Image Generation Policy

The proposed image augmentation policy via synthetic image generation exists as a step that exists prior to any traditional image augmentation steps, generating synthetic images that are passed along to the subsequent image transformations that exist downstream. Given a proportion of original training images to utilize for label i— denoted as X_i—and the desired proportion of synthetic images to augment the training data for label i— denoted as Y_i—we combine both synthetic and original images in order to create a mixed train dataset used to train the image classification model. In practice, X_i would almost always be 100% as you generally want to train on available data you have; however, we test different values of X_i in order to test the efficacy of our image generation policy across a range of possible dataset sizes.

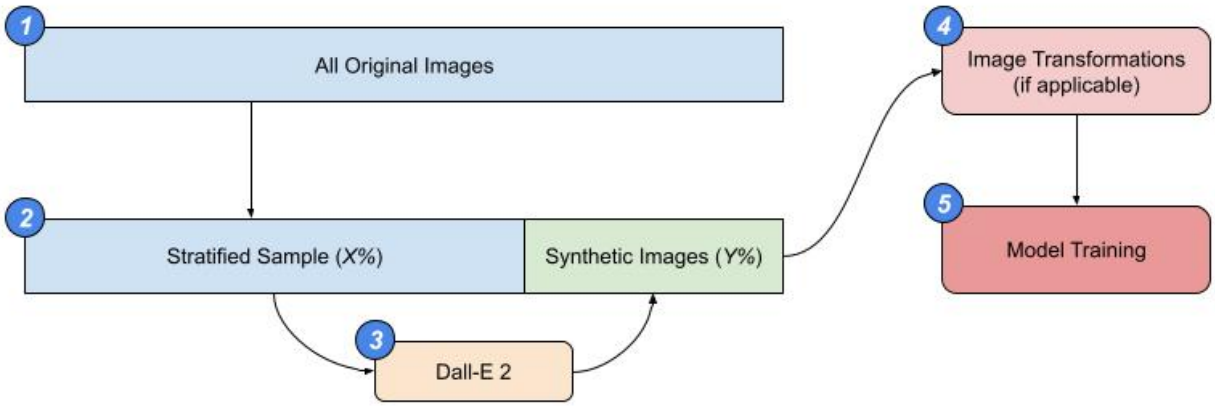


Figure 6.2: Illustration of the image generation policy

The figure above illustrates our image generation policy in order to generate our mixed training set (i.e., containing both original and synthetic images)

1. First, we take a random sample of our original images
2. The sample is sent to Dall-E 2 via OpenAI's API in order to generate an image variation
3. Image transformations are applied to our entire mixed training set (if applicable)
4. Mixed training set, post transformations/augmentation, is used to train the image classification model

CHAPTER 7

Methodology and Results

7.1 Methodology

To test the effects that synthetic images have on image classification performance, we tested a variety of model architectures paired with differing image augmentation protocols. We also tested protocols that involved synthetic image augmentation for all classes in our original dataset and ones that involve augmentation for select classes only. Each combination is trained 20 times on independent samples in order to get a distribution of results and reported results are based on the median of the distribution of measured performances.

Scenarios tested were:

- Training on 100% of available training data
- Training on 100% of available training data and an additional 20% consisting of synthetic images
- Training on 80% of available training data
- Training on 80% of available training data and an additional 20% consisting of synthetic images images

To avoid overfitting our models, after each training epoch the performance of the model is tested against a validation data set that is held out. If the model fails to improve (i.e., reduce the loss function) against the validation data set for 2 consecutive epochs, that particular training run is ended and the model performance is logged.

7.2 Basic CNN Results

Cross Entropy Loss				
Scenario		No Augmentation	Random Horizontal Augment	AutoAugment
1	100% Real / 0% Synthetic	0.113512	0.173663	0.15266
2	80% Real / 0% Synthetic	0.23912	0.270616	0.274309
3	100% Real / 20% Synthetic (Mild & Moderate Demented Only)	0.103296	0.169054	0.137484
4	80% Real / 20% Synthetic (Mild & Moderate Demented Only)	0.213611	0.247438	0.242083

The basic image augmentation and AutoAugment protocols resulted in a slight degradation of model performance across all training scenarios, illustrating that simply applying image augmentation is not always recommended when training image classification models.

Similarly, synthetic image augmentation applied equally across all image classes results in a slight increase in cross entropy loss. This is likely due to Dall-E-2 struggling to produce non-demented images with sufficient fidelity.

However, models trained on datasets with image augmentation applied to only the 2 most demented image classes (labeled as mildly and moderately demented) performed better than those trained on synthetic images alone. Comparing scenarios 1 and 3, synthetic image augmentation to specific classes resulted in a 9.0% decrease in cross entropy loss when paired with the best performing traditional image augmentation protocol (no augmentation), reducing loss from from 11.4% to 10.3%. This holds true for scenarios 2 and 4 as well, which trained models on 80% of the available real training images. The introduction of synthetic images reduced error by 10.7%, from 23.9% to 21.4%.

7.3 ResNet Results

Cross Entropy Loss				
Scenario		No Augmentation	Random Horizontal Augment	AutoAugment
1	100% Real / 0% Synthetic	0.126419	0.216701	0.954748
2	80% Real / 0% Synthetic	...	...	...
3	100% Real / 20% Synthetic (Mild & Moderate Demented Only)	0.119202	0.175533	0.592514
4	80% Real / 20% Synthetic (Mild & Moderate Demented Only)	0.209497	0.259539	0.785979

ResNet models reacted to synthetic images similarly to our basic CNN architecture, with findings from the ResNet model training scenarios aligning directionally with the results found when using the CNN architecture. Comparing scenarios 1 and 3, synthetic image augmentation to specific classes resulted in a 5.7% decrease in cross entropy loss when paired with the best performing traditional image augmentation protocol (no augmentation), reducing loss from from 12.6% to 11.9%. The same if true for scenarios 2 and 4 with synthetic images reducing error by XX.X%, from XX.X% to XX.X%.

7.4 ViT Results

Cross Entropy Loss				
Scenario		No Augmentation	Random Horizontal Augment	AutoAugment
1	100% Real / 0% Synthetic	...	...	...
2	80% Real / 0% Synthetic	...	...	...
3	100% Real / 20% Synthetic (Mild & Moderate Demented Only)	...	...	...
4	80% Real / 20% Synthetic (Mild & Moderate Demented Only)	...	...	...

CHAPTER 8

Conclusion

Models that were augmented with synthetic images across all classes often showed a degradation in performance. For our best performing model architecture (a basic convolutional neural network), a model trained on 80% of all available original training images and augmenting all classes with 20% synthetic images did not perform as well as a model trained on 100% of all available original training images. Moreover, it also failed to outperform models trained on 80% of available training images and no synthetic image augmentation, indicating that augmenting all classes with synthetic images is not necessarily a universal solution and may actually hurt model performance.

In contrast, models augmenting only the mildly and moderately demented classes with synthetic images had increased performance. For basic convolutional networks, cross entropy loss reduced by 9.0% when utilizing 100% of the available training images and augmenting with synthetic images by 20% versus training on 100% of training data alone. This may be due to the quality of images generated by dall-e-2, which may be better suited to generate images of brains with larger signs of atrophy which is associated with more severe cases of dementia.

Furthermore, traditional image augmentation methods proved to be ineffective—even ones designed to programmatically choose optimal augmentation methods. We found that synthetic image augmentation of select classes resulted in a model that outperformed ones trained using traditional image augmentation—including automated protocols such as AutoAugment.

{INSERT PARAGRAPH ABOUT ViT RESULTS HERE}

CHAPTER 9

Additional Considerations

The goal of this paper was to measure the effects synthetic image augmentation and how they compared to traditional image augmentation methods. However, due to its limited scope, there are a few additional considerations that should be noted.

While this paper tests a variety of model architectures, the purpose was to observe how synthetic image augmentation interacts with both a convolutional architecture and an attention-based architecture. Model hyperparameter tuning was not taken into account, and model parameters such as learning rate, batch size, and the number of layers were not adjusted. As a result, a model's performance should only be compared against other models of that same architecture. Further model tuning may improve any given architecture's performance, but our focus for this paper remains on the relative change in performance within each architecture when comparing different image augmentation protocols.

Another consideration is the choice of generative model chosen for the scope of this paper. While there are many image generation model available, OpenAI's Dall-E-2 was chosen for the convenience that the API provided. However, there are a variety of both open source and closed source generative models that may prove to be better suited for synthetic image augmentation tasks.

Furthermore, while synthetic image augmentation has the potential to improve model performance, test results show that applying augmentation to all classes may actually hurt performance. When applying to the most severely demented classes, however, model performance improved significantly. Additional reearch into applying synthetic image augmenta-

tion programmatically may be beneficial in developing an automated protocol, rather than manually choosing which classes to augment with synthetic images.

REFERENCES

[AI24] Albumentations AI. "What is image augmentation and how it can improve the performance of deep neural networks.", 2024. https://albumentations.ai/docs/introduction/image/$_{a}ugmentation/(LastAccessed: 2023-03-20)$.