Class Activation and Attention Models for Face Classification

Yu Chen

September 4, 2019

Abstract

In traditional deep learning models, each layer of the neural network architecture attempts to learn, or "generalize", increasingly higher and higher level abstractions of the original feature set. In computer vision and image recognition, for instance, the first few layers of a convolutional neural network (CNN) might identify edges, lines, or shifts in directions, while later layers begin to generalize shapes, and ultimately real-world mapped objects like a nose, a vehicle, person, etc. When a neural network is exposed to different classification or regression problems, only certain sections or groups of features will be truly relevant. For instance, in natural language processing, RNN and LSTM models have demonstrated that generating the next word in a sequence of words may include features from several words, or even several sentences away. In this paper, I attempt to replicate the work of Vaswani et. al. in *Attention Is All You Need* [4] as well as Jacob Gil's studies with Class Activation Maps, utilizing a dataset of over 2,800 face images from approximately 200 individuals initially collected by the Artificial Intelligence Laboratory of FEI in Sao Paulo, Brazil [5]. I aim to train a computer vision model on several classification problems - gender, subject identity, age category - to identify pattern shifts in attention weighting and class activations based upon the classification task at hand.

1 Code and Dataset Access

The source code for this analysis is publicly available as a Github repository. Instructions for replicating findings are available in the README.md file in the root directory. Please contact me for access to the raw image S3 bucket, as the default access control is private, per the distribution license of Dr. Carlos Eduardo Thomaz.

2 Introduction

In Attention Is All You Need, the authors introduce a new model architecture that does not rely upon recurrent neural networks, instead utilizing a self-attention layer to compute latent representations of the input features. These architectures have been almost always designed for sequence-to-sequence problems in mind. There are two main

flavors of attention architectures within the industry: **Transformers** and **Encoder-Decoders**. Attention mechanisms can be implemented in various ways, but output a score that reflects how much emphasis, or "attention" should be focused on a single element of a sequence. Within the context of images, we can think of an image as a two-dimensional sequence, for instance.

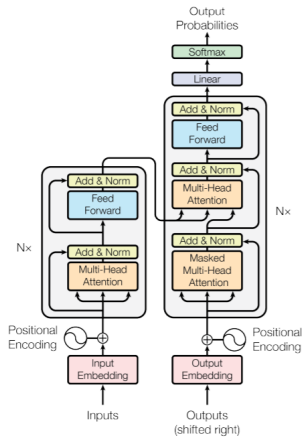

Figure 1: The original transformer architecture proposed by Vaswani et. al, consisting of parallel encoder and decoder stacks, each with $N = 6$ identical layers. In each layer, the input embeddings (typically latent space word embeddings such as word2vec or GloVe) are feed through a self-attention layer prior to its traditional feedforward layers.

3 Prior Work

3.0.1 Self-Attention

Significant resources for self-attention models have historically been directed towards natural language processsing use cases, and in particular implemented within recurrent neural network (RNN), gated recurrent units (GRUs), and long short-term memory units (LSTMs). However, traditionally, attention models have been confined mostly to text data, since it is difficult to properly encode positions across more than one dimension (which is necessary for image recognition and computer vision tasks).

The traditional steps associated with self-attention computation are

1. Select a dimension for the attention feature projection - we are projecting the input feature space into a smaller dimension, so we need to select some $d < D$, where D is the dimensionality of original input sequence. Within the context of natural language processing, this would typically be the embedding dimensionality (ie. 100, or 150).

2. Randomly initialize three separate weight matrices - W_q, W_k, and W_v, with each $\in \mathcal{R}^{D \times d}$ (original number of input dimensions by projected dimensions).

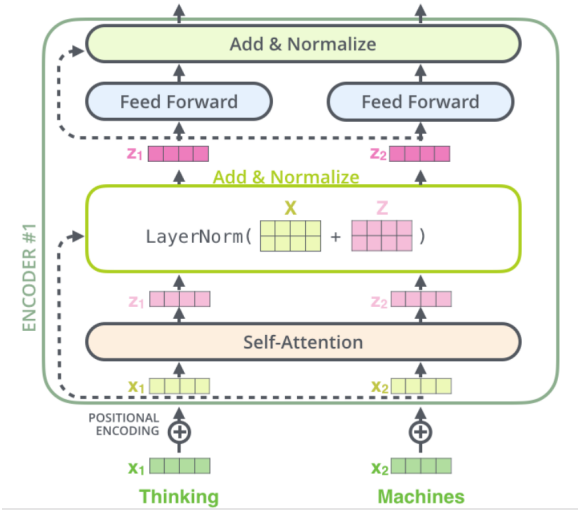

Figure 2: A more detailed architecture of the encoder from Jay Alammar's The Illustrated Transformer blog article [1]. Since the Transformer architecture declines to utilize recurrence, it inherently loses its ability to capture the semantic meaning of word ordering. The positional encoding layer is an additional feature layer intended to help the model account for the positional location of a feature (usually a word) within the context of that sequence.

3. Receive as input a matrix $X \in \mathcal{R}^{N \times D}$ (N input samples, each of D dimensions)

4. Multiply X by the query, key, and value weight matrices to generate the query, key, and value matrices:

$$Q = X \times W_q \tag{1}$$

$$K = X \times W_k \tag{2}$$

$$V = X \times W_v \tag{3}$$

   Each of these new computed matrices (Q, K, and V) are of shape $\mathcal{R}^{N \times d}$

5. Compute a scaling factor, $\sqrt{d_k}$ that is used to stabilize the gradients during the weight update step (backpropagation). d_k is the number of dimensions used in projected dimension for the keys. So if $d_k = 64$, then the scaling factor would be 8.

6. Compute a final score matrix Z:

$$Z = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{4}$$

   Z will be of shape $\mathcal{R}^{N \times d}$. Notice the output of the self-attention layer is not of $\mathcal{R}^{N \times D}$ (its dimensionality is reduced). We will discuss solutions to recover the initial dimensionality in the Multihead Attention section.

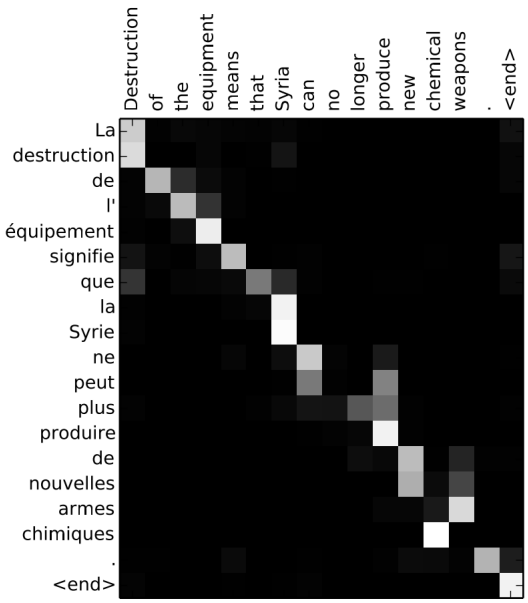

Figure 3: A visualized heatmap of a sequence-to-sequence language translation model, with an input French sentence's tokens as rows and the sentence's English equivalent as columns. This is an example of a many-to-many classification problem, as at each sequence step, the model must output a softmax prediction of the most likely English translation. The lighter the color, the more emphasis a specific word in the past was given by the model when determining the latent semantic representation and subsequent English translation. Note that the length of the French input token sequence differs from that of the English translation. [1]

3.0.2 Multiheaded Attention Augmentation

The particular implementation of self-attention I selected for this study was the **Attention Augmented Convolution Network** (Google Brain). This is a recent advancement on an established pattern called **Multihead Attention**, which computes attention scores in parallel using multiple sets of query, key, and value matrices. Therefore, using a 4-headed attention model, we would have 4 separate key matrices: W_{k1}, W_{k2}, W_{k3}, and W_{k4}, and long with 4 separate query matrices and 4 separate value matrices.

3.0.3 Permutation Equivariance and Translation Invariance

The authors of the MHA augmentation model have noted that, by itself, self-attention models are completely permutation equivariant. [3] This means they satisfy the equality

$$MHA(\pi(X)) = \pi(MHA(X)) \tag{5}$$

using permutation function of the pixels in the input image π. This is not a desired quality, as it means that the model loses all concept of "position", which is fundamental

when receiving image data. Indeed, without any sort of positional encoding, the input image could simply be fed in as a $H \times W$ single dimension vector.

In many natural language processing sequence-to-sequence models, this challenge is overcome by encoding a learned positional vector that is summed with the input embedding. Typically, this positional encoding is computed by incrementing each dimension of the input tensor with a sinusoid function with different frequency and phase parameters, allowing the attention to learn absolute and relative positions. The authors of MHA augmentation introduce a form of **relative position encodings** to encode the distance (and relationship) between two different pixels i and j: Here, Q is the original

$$O_h = \text{Softmax}\left(\frac{QK^T + S_H^{rel} + S_W^{rel}}{\sqrt{d_k^h}}\right)V$$

query matrix, K is the key matrix, V is the value matrix, and S_H^{rel} and S_W^{rel} are the learned relative position weight matrices that encode the relationship between a pixel i and another pixel j.

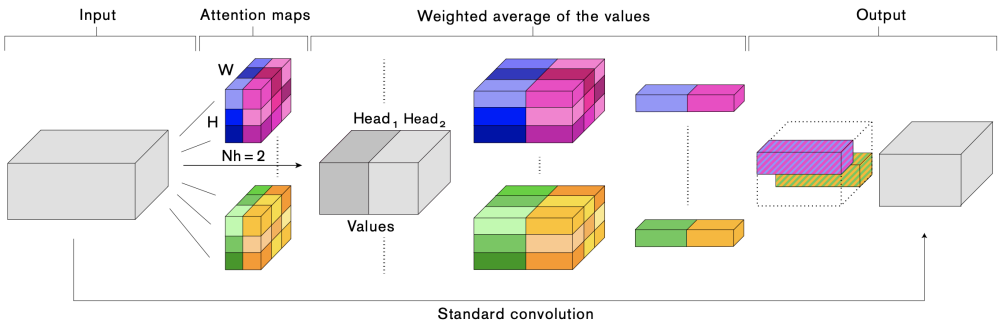

Figure 4: A high-level workflow using Multi-Head Attention Augmentation. The original input is fed in twice - once to a traditional vanilla convolutional neural network, which outputs a feature map (highlighted in green and orange). In parallel, attention maps compute weighted average value matrices using self-attention using multi-head attention workflows to parallelize the computation. The output of these two maps are concatenated into the final output.

3.1 Class Activation Maps

An alternative form of explainability to attention models for convolutional neural network architectures is are Class Activation Maps (CAM). These maps are built upon a unique aggregation layer called **global average pooling**, which outputs the average of the feature map for each filter. In the MNIST toy dataset implementation of CAM, the final convolution layer's output is of shape $24 \times 24 \times 32$. If $f_k(x, y)$ represents the value

of the k-th feature map at position x, y, then the GAP function can be written as

$$F^k = \sum_{x,y} f_k(x, y) \tag{6}$$

In our MNIST toy dataset example, the GAP layer would receive as input $24 \times 24 \times 32$ and output a vector of size 32 [8]. In the traditional classification step, the final input to the softmax function would be $S_c = \sum_k w_c^k F^k$ and the final probability outputted for class c is

$$P(c) = \frac{exp(S_c)}{\sum_c^C exp(S_c)} \tag{7}$$

Here, we see that each value of w_c^k corresponds with an "importance" score for the k-th feature output. By rewriting our definition for S_c, we get the basis for in intuition of class activation maps:

$$S_c = \sum_k w_c^k \sum_{x,y} f_k(x, y) \tag{8}$$

We can move the outer summation inside the inner:

$$S_c = \sum_{x,y} \sum_k w_c^k f_k(x, y) \tag{9}$$

Now, we have a definition of a class activation map, M_c:

$$M_c(x, y) = \sum_k w_c^k f_k(x, y) \tag{10}$$

Now, we see that the final score for a specific class c is

$$S_c = \sum_{x,y} M_c(x, y) \tag{11}$$

We have a one-to-one mapping, M, between the pixels of original input features and their "importance" in contributing to the classification of the class target.

4 Experimental Design

4.1 Datasets Used

4.1.1 MNIST

An initial toy dataset was selected to prototype the models developed within this paper. The reasoning for this was as follows: if the model could not achieve functional performance on a known, "easy" problem, then it would be a waste of computational resources to attempt classification on a significantly more challenging and resource-intensive problem such as gender recognition.

As such, I selected famous MNIST dataset, which contains handwritten digit images, to prototype my models off of. These images were already centered, scaled, and transformed into black or white, making the initial task of loading the dataset into memory and performing training significantly easier.

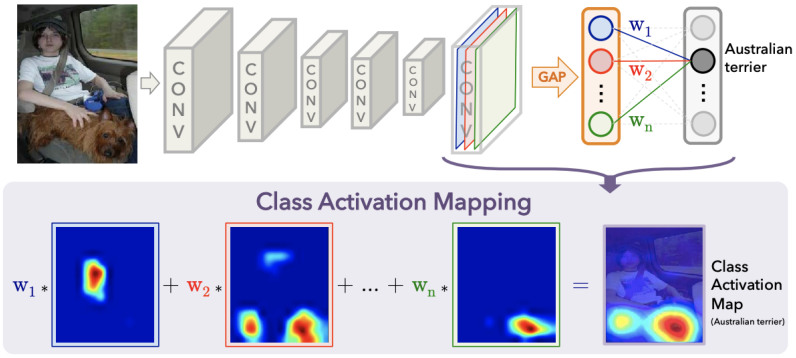

Figure 5: Class activation mapping for a classification problem of identifying if a dog is present in the image. Notice that the model performs the traditional convolution layer operations but then aggregates the final outputted feature maps using a **global average pooling** layer. This layer is then mapped back into the original input image dimensions, where areas of high activation correspond with relevant regional feature spaces for classifying the target.

4.1.2 FEI

The dataset selected for this experiment is from Artificial Intelligence Laboratory of FEI in Sao Paulo, Brazil, available publically on the FEI website, and consists of approximately 2,000 images, each 640 x 480 pixels in size, of approximately 200 subjects, 100 male and 100 female, between the ages of 19 and 40 years old. Each subject was featured against a uniform, white background with profile rotation of up to 180 degrees. Each subject was photographed between 10 and 12 times, at various angles, and also with a variety of facial gestures. Certain images were excluded from the final dataset if they were too dark or grainy- approximately 100 images were dropped for image quality reasons.

4.1.3 Infrastructure and Hardware

All images are stored in an AWS S3 bucket called $fei-faces-images$. These images are publicly available for further research by the data science / biometrics community.

A specialized g3.4xlarge GPU-optimized AWS EC2 (Elastic Compute Cloud) instance operating was provisioned for these experiments. Given the significant memory and computational requirements required to train the VGG-based model, an AWS Deep Learning AMI operating on Ubuntu was selected to meet infrastructure requirements. The $data-preprocessing$ module inside the Github repository associated with this paper contains the logic required to properly tag and upload images files to the S3 bucket.

The actual model development was performed within a virtual environment using Python 3.6 and Tensorflow 1.4, as well as a Keras abstraction-layer wrapper. The notebooks were served remotely, with developers (me) first SSH-ing into the EC2 instance

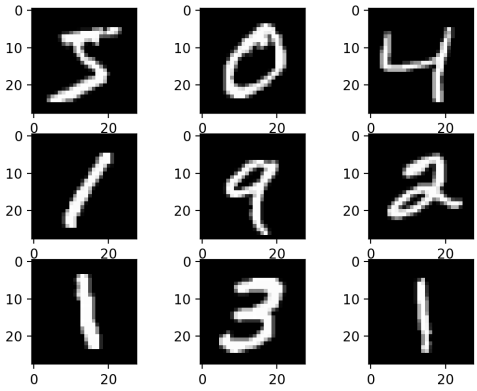

Figure 6: Examples of handwritten digits that comprise the MNIST dataset. [7]

Figure 7: An example of an image set for a particular subject [5]. Each subject's face was captured at a variety of rotational angles. Notice the final image is obscured significantly with shadows.

to start the Jupyter notebook server process, and then logging in via an access token to the HTTP endpoint.

A key note is that training multi-headed self-attention (MHA) models is extremely compute and memory intensive. In order to reduce the memory load required, significant image resizing on the original face images was performed. In addition, early stopping and extremely small batch sizes were used. As a result, the models were trained only for a small fraction of their optimal training epochs.Therefore, the holdout test performance metrics reported in this paper almost certainly underestimate the true performance of an MHA model.

4.1.4 Model Evaluation Metrics and Objective Function

We chose to use a generic evaluation metric of accuracy, although the objective function (by which the gradients for the deep learning model's backpropagation updates were based) used **binary cross-entropy**. Cross-entropy is an ideal measure for classification models that output a probability value (a number between 0 and 1), and can be defined mathematically as

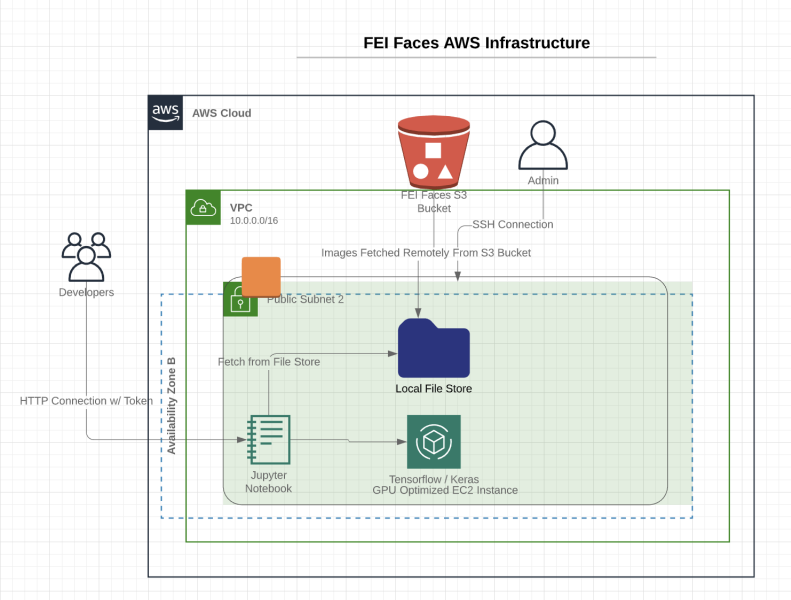

Figure 8: Schematic of AWS infrastructure. Images are hosted on AWS S3 object store and fetched at runtime to the local compute instance. Developers connect remotely to the Jupyter notebook interface via the public DNS HTTP endpoint and a private token that is provided by the admin. The admin starts the Jupyter server and publishes the access token.

$$-(ylog(p) + (1-y)log(1-p)) \tag{12}$$

For future studies, we also recommend the **Cohen's kappa score**, since it takes into account the "lift" a model provides over blind guessing within classification problems. For instance, baseline model performance should be pinned against the accuracy obtained by simply predicting the most frequent class sample (mode). For instance, if 90% of images in a dataset are of men, then a model with an accuracy of 91% for gender classification may actually be performing barely above random guessing. Thus, we select **Cohen's kappa score** to quantify the lift above this baseline classification, where the score k ranges from -1 to 1, and is defined as

$$k = \frac{p_o - p_e}{1 - p_e} \tag{13}$$

Here p_o is the empirical probability of agreement on the target class (the observed agreement ratio), and p_e is the expected agreement with two models assigning random labels.

4.2 Methodology and Approach

I began my experiments by first implementing vanilla CNN architectures, both on the MNIST datasets and then on the main FEI dataset. Each image was initially classified manually by the relevant target (in other words, in order to tag gender, I placed all male images in one folder and female images in another, and then wrote a Python script to autogenerate a JSON file containing these mappings that was stored in an S3 bucket). Each tests begins similarly:

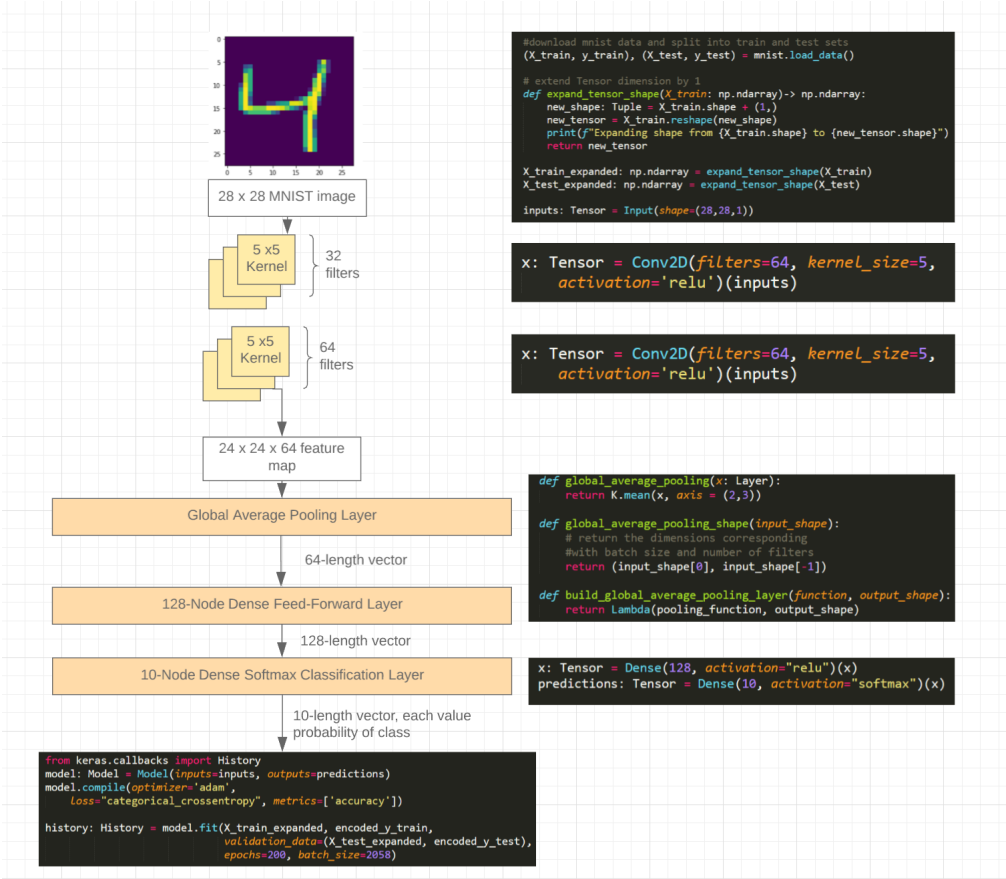

Figure 9: The initial model architecture on the MNIST dataset, with a Global Average Pooling layer implementation of CAM (class-activation heatmaps. The output of the model is a 10-element vector that corresponds with the predicted probabilities of each image class.

1. The metadata JSON file containing the target classes for each image is downloaded by the test scripts.

2. Each image file is downloaded onto the local working directory of the machine running the model, and then subsequently loaded into memory.

3. A resizing factor of 50% - 75% is applied to reduce the size of the image from 640×480 pixels to a far small value, allowing for more efficient training and computation.

4. A two-dimensional Numpy array is created with one-hot encoded target classes.

5. The architecture of the deep learning model is constructed as a directed acyclic graph computation using Keras

6. The model is compiled and fit to the training data. At each iteration, the training data is permuted, with approximately 30% of samples being held out as a validation set.

7. After the model finishes training, the weight matrices from specific layers are extracted and used to construct the heatmap overlay over the original image in order to detect what parts of the image the model spent the most time focusing on

A **batch normalization layer** was added to the end of the deep learning architecture as a way to reduce overfitting and variance in the outputs of the layers. Normalization essentially constrains outputs to be between certain bounds, allowing the final prediction to take on more reasonable values (as opposed to converging to either 0 or 1).

4.2.1 Implementation of Class Activation Maps

Class Activation Map models were implemented using tutorials and source code based off of Jacob Gil's *Class activation maps in Keras for visualizing where deep learning networks pay attention* blog article. [2] I was able to more or less follow Jacob's implementation exactly, based upon the original 2016 MIT paper.

An example output of the class activation maps on the MNIST toy dataset is shown in Figure 10. Readers will notice that when classifying the handwritten image of the number 2, the activation heatmaps differ from class to class (ie. the model looks at slightly different elements of the image when checking if the image is of a 5, or a 6.

4.2.2 Implementation of Multi-Headed Augmented Attention

The architecture for the Augmented Attention model is based primarily on the concept of the Multi-Head Attention problem. I directly borrowed several classes from an implementation by Somshubra Majumdar's Github repository. [**?**]. I needed to make several modification to the original algorithm in order for MHA augmentation to work properly in conjunction with VGG16:

- The VGG model that comes prepackaged by Keras expects two different input streams, one for the VGG layer and one for the overall model. Although this is essentially the same input, I had to unpack Keras' implementation of VGG16 by building the layers from scratch, although much of this was accomplished by borrowing code from the Keras official applications repository.

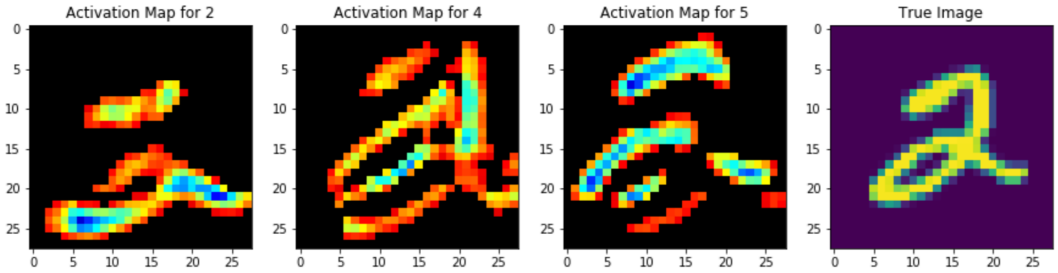

Figure 10: The initial model architecture on the MNIST dataset, with a Global Average Pooling layer implementation of CAM (class-activation heatmaps. The output of the model is a 10-element vector that corresponds with the predicted probabilities of each image class.

- I added the Multi-Head augmented attention layer immediately after the VGG model received input. The final output of this augmented layer was of shape $H \times W \times 12$, whereas the original image input tensor is of shape $H \times W \times 3$. Therefore, I added a reshape layer to the model such that the input image to the VGG layer was actually $2H \times 2W \times 3$.
- Again, I froze the last 5 layers of the VGG model, to reduce the number of trained parameters from approximately 14.8 million to 7.1 million.

5 Results

5.0.1 Model Performances on MNIST Toy Dataset

1. **Vanilla CNN Model A**: A 4-layer traditional CNN model, with an initial 128-node dense layer, two 2D convolutional layers with filter sizes of 64 and 32 respectively, and a final dense layer for softmax classification.
2. **Vanilla CNN Model B**: A simplified 3-layer CNN model, identical to **Vanilla CNN Model A**, but with the initial 128-node dense layer removed.
3. **Max Pool Deep CNN Model w/ CAM** : A CNN with two layers of 2D convolutional layers of 32 filters by 64 filters of kernel size 5×5, separated by a Max Pooling with stride length of 2×2. These are aggregated with a global average pooling layer that is then stacked a 128-node dense feed forward layer and final dense layer for softmax classification.

Table 1: MNIST Model Performance Results

Model -	**Training** Accuracy	**Validation** Accuracy
Vanilla CNN Model A	96.7%	97.1%
Vanilla CNN Model B	94.5%	96.6%
Max Pool Deep CNN Model w/ CAM	99.6%	98.9%

5.1 FEI Dataset

5.1.1 Overall Results

There were a variety of different model architectures that were tested for the FEI Faces Dataset. In general, most of the architectures derived from the VGG16 architecture, which is a convolutional neural network that has achieved industry-standard accuracies of 92.7% on ImageNet, which is a common benchmark dataset for image recognition that contains over 14 million images and 1000 classes. The VGG architecture consists of 5 (2-layer deep) sections of convolution, culminated with a max pooling layer with a 2×2 pixel window and stride of 2, and ultimately three fully connected dense feedforward layers.

Because the FEI faces dataset is significantly more complex than the MNIST dataset, I only had time and financial resources for a few model runs. Therefore, I selected only two architectures, both relatively basic and clearly with opportunity to tune, in order to focus on getting viable results. For the FEI Faces gender classification task, several architectures utilized for both multi-headed attention (MHA) and class activation maps (CAM) models:

1. **MHA VGG model**: An augmented attention layer followed by the traditional VGG16 architecture and two feed-forward layers with a batch normalization layer
2. **CAM VGG model**: VGG16 architecture, followed by a batch normalization layer and two feed-forward layers. Coincidentally, the VGG16 architecture already has a built-in global aggregation layer (either average or max), so I simply took the outputs from this layer and swapped out the default max aggregation layer for an averaging layer.

Table 2: FEI Faces Model Gender Classification Performance Results

Model –	**Training** Accuracy	**Validation** Accuracy
VGG Model with Multiple Head Attention	77%	85%[1]
VGG16 Max Pool Deep CNN Model w/ CAM	100%	96.2%

5.1.2 Class Activation Map Results When Classifying Gender

In general, the Class Activation Map (CAM) models focused particular attention on the neck and upper body portions of a subject's image.

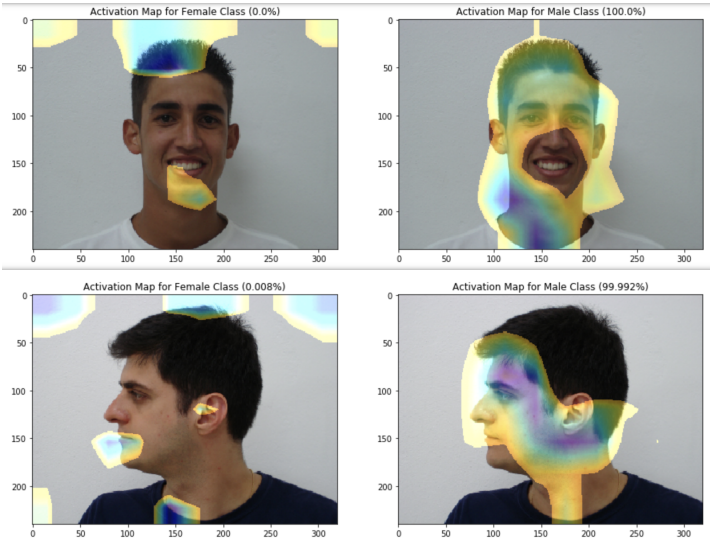

Figure 11: Two male subjects, both correctly classified as male by the model, representing pictures taken from different angles. For the top subject, the model focused on his neck region and hair style in making the male classification. One prominent exclusion was the mouth, and in particular the smiling gesture. Since each subject was photographed over 10 times, with certain images designated to be smiling, the model likely noticed that smiles did not correlate one way or the other with a particular gender.

From the class activation maps, a few common themes are clear:

1. Earrings or earlobe decorations are used by the model to disqualify subjects for male classification
2. Exposed shoulder or neck skin, as well as bangs for frontal images is used for female classification
3. The shape and texture of the eye and eyebrow regions are used as distinguishing features for male classification
4. When presented images of a subject facing the camera at an angle, the model looks for particular jawline and sideburn-area patterns to classify the subject as male

[1] I did not completely finish training this model, given its computational complexity and extremely large memory requirements. Thus, you can see that it would likely have performed much better if more epochs were added to its training workflow.

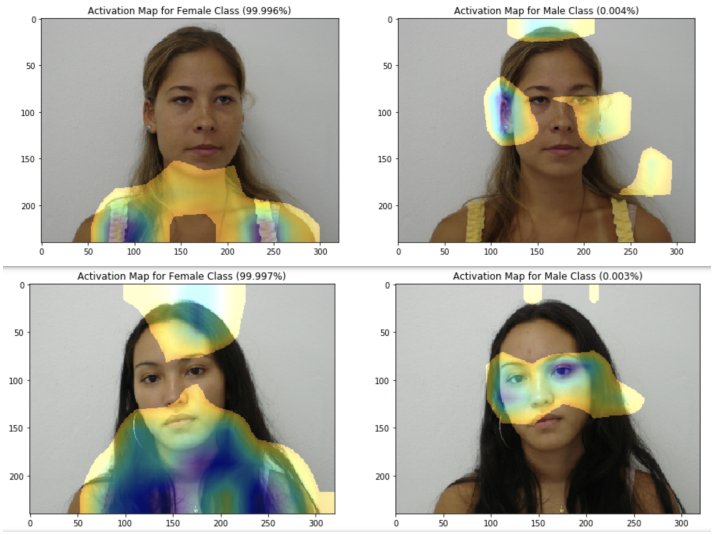

Figure 12: Two female subjects that were correctly classified by the model. Notice the significant attention paid to the subjects' tank top straps and chin in classifying them within the female classes- this is a common theme in the model - potential female subjects tend to wear clothes that expose a greater proportion of skin for the shoulders, neck, and upper body.

5. The model is most ambiguous about female subjects, particularly when they are wearing "non-feminine" attire such as t-shirts, or sweatshirts that completely cover their upper torso.

Out of the 700+ images in the FEI validation set, only three images were extremely ambiguous, which is defined as a classification with a probability of less than 60%. Notice that the model focused in particular on the subject's sideburn region when looking for male characteristics, while the neck, top of the head, and collarbone regions were the areas of interest for female features. Moreover, in both images, the model struggled to find any elements of hair (bangs, ponytails) and clothing (tank top straps, exposed shoulders, etc.) that are traditionally associated with female fashion.

5.1.3 Multi-Head Attention Results When Classifying Gender

When inspecting results from the Multi-Head Augmented Attention (MHA), readers will notice a severe reduction in the overall quality of the images - many of them will appear quite grainy and low-resolution. Due to the extremely expensive memory requirements of the MHA model, I was forced to significantly reduce the size of the original image (by a factor of 8) in order to avoid Out of Memory issues while allocating tensors within the underlying Tensorflow engine. In general, MHA augmentation focused its attention on the particular hair styles of subjects. Surprisingly, very little of the face was used in

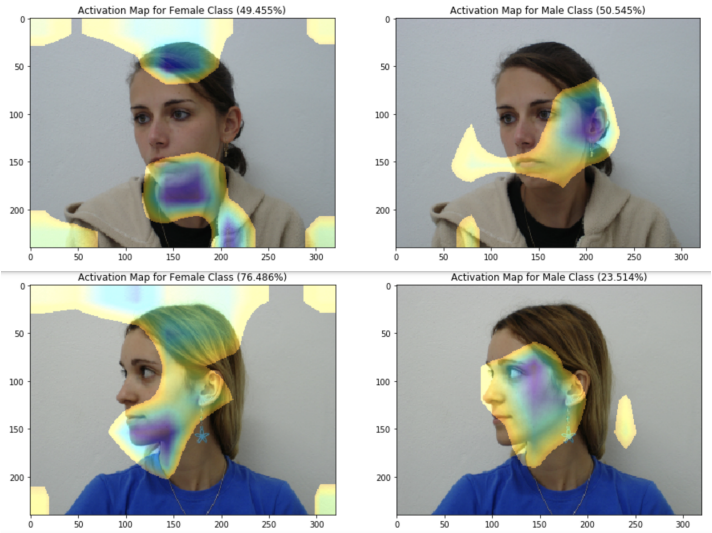

Figure 13: Two relatively ambiguous target classifications from the holdout set for the VGG-backed model using the CAM model. The top image was incorrectly classified as Male. Notice that the model focused in particular on the subject's sideburn region when looking for male characteristics, while the neck, top of the head, and collarbone regions were the areas of interest for female features. Moreover, in both images, the model struggled to find any elements of hair (bangs, ponytails) and clothing (tank top straps, exposed shoulders, etc.) that are traditionally associated with female fashion.

determining gender, except for certain regions around the eyes and mouth. The most prominent region of skin utilized was the neck.

Similar to the CAM models, the MHA models also tended to focus heavily on the subject's neck region, particularly for men. It appears that given the grainy resolution, one of the few discriminating features between male and female subjects were hair patterns - the images has lost too much resolution during the resizing operation to highlight subtle differences in facial skin texture.

5.2 Discussion and Key Takeaways

There were several key takeaways from my work with CNN architectures:

5.2.1 Learning Rate

The learning rate of the CNN plays an extremely large role in optimizing performance of the model. Since the FEI faces dataset is split between training and validation sets, we obtain approximately 719 validation samples that are held out during the training phase. If the learning rate is set too high, the model validation fluctuates wildly, and does not make any progress from epoch to epoch. We have found that any for learning rate above 0.00001 yields unstable validation performance, however, a learning rate of $1e-6$

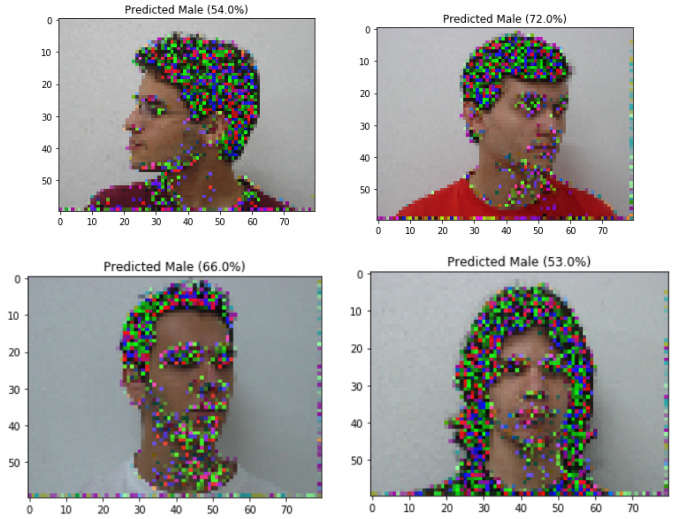

Figure 14: Three subjects correctly classified as Male by the Multi-Head Augmented Attention model. In both cases, attention was paid in particular to the center neck region (likely presence of neck muscles or Adam's apple) as well as the hair styles. Notice also the third subject features significant attention on areas of his face where facial hair tend to grow (sideburns, chin, cheeks).

does indeed ensure that the model will eventually converge towards a more stable (local, but hopefully global) minimum, but does significantly increase the required number of epochs.

5.2.2 Freezing Layers

A core reason for VGG16's exceptional performance on image recognition tasks is its complex architecture. Even its most basic version (preloaded with Keras) features approximately 14,714,688 weight and filter parameters that require updates each training batch iteration. In our implementation, I have frozen the last 5 layers of the VGG16 model, such that we reduce the number of trained parameters by approximately 50%, to 7,080,450 parameters updated each batch.

5.2.3 Cultural Norms

It was interesting to peek into the "logic" the face classification models were using to classify a person's gender. Many of the patterns and feature regions appeared to reflect ingrained cultural norms of femininity and masculinity. For instance, clearly female clothing patterns were encoded into both the CAM and MHA augmented models. Both models also focused specifically on hairstyles as well as a primary region for determining gender. A more robust model would be able to correctly classify males / females even when they are wearing wigs, or gender-neutral clothing.

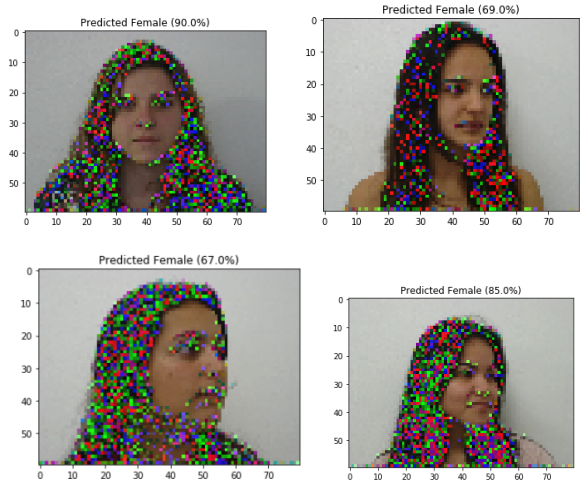

Figure 15: Three subjects correctly classified as Female by the MHA model. Notice that for female subjects, the model focuses particular attention on hairstyles, eyebrows, and the type of clothing worn on the upper torso. This is not altogether unexpected, given the low resolution of most of these pictures, there is not much on the face itself that would indicate differences between males and females.

5.3 Opportunities for Future Work

5.3.1 Other Facial Expressions

I have already tagged and classified all 2,600 images by smiling/not smiling in addition to gender. Arguably, this is a more challenging problem, as an average human being is typically able to discern the gender of most individuals relatively quickly, but there can be significant ambiguity as to whether a subject is truly smiling - this information is frequently dependent upon the subject's personality, the context of the image, etc. Therefore, one further area of research is to identify the features the CAM and MHA augmentation models focus on when classifying smiles. Of course, I expect that a significant amount of attention will be devoted to the month region, for the presence of teeth, but I would also expect focus on the shape of the cheeks and eyebrows (simply showing teeth does not mean someone is smiling).

5.3.2 Gradient-Based Class Activation Maps

Jacob Gil notes in his blog that the CAM model implementation I used for this paper requires altering the fundamental structure of the model itself. For instance, a GAP (global average pooling) layer is required at some point, preferably near the final dense output layer, to properly map the input pixels to a class activation map. Gil also introduced a Gradient-Based CAM model that does not require modification to existing neural network architectures. If properly implemented, this model could prove vital in injecting a formerly complete black box machine learning model with human interpretability.

5.3.3 Other Attention Architectures

I tried only one implementation of MHA, given time constraints. However, significantly other design patterns remain unexplored, especially ones that more adequately trade off model performance with overall training efficiency and memory usage. The authors of the MHA paper highlighted several key steps they took to mitigate memory bottlenecks that I was not able to implement.

References

[1] Jay Alammar. *The Illustrated Transformer*. http://jalammar.github.io/illustrated-transformer. Accessed September 3rd, 2019.

[2] Jacob Gil. *Class activation maps in Keras for visualizing where deep learning networks pay attention*. https://jacobgil.github.io/deeplearning/class-activation-maps. Accessed August 22nd, 2019.

[3] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, Quoc V. Le. Google Brain. Submitted on April 22nd, 2019. Published August 14th, 2019. *Attention Augmented Convolutional Networks*. https://arxiv.org/pdf/1904.09925.pdf. Accessed August 24th, 2019.

[4] Ashish Vaswani, et. al. Le.. Submitted on June 12th, 2017. *Attention Is All You Need*. https://arxiv.org/pdf/1706.03762.pdf. Accessed August 21st, 2019.

[5] FEI Faces Dataset. Carlos Eduardo Thomaz, Image Processing Laboratory, Department of Electrical Engineering. Centro Universitario da FEI, Sao Bernardo do Campo, Sao Paulo, Brazil. et. al. Le. *Attention Is All You Need*. https://fei.edu.br/ cet/facedatabase.html. Accessed August 1st, 2019.

[6] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba. *Learning Deep Features for Discriminative Localization*. http://cnnlocalization.csail.mit.edu/. Published 2016.

[7] Jason Brownlee. *How to Develop a CNN for MNIST Handwritten Digit Classification*. https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-from-scratch-for-mnist-handwritten-digit-classification/. Published May 8th, 2019. Accessed September 4th, 2019.

[8] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, Antonio Torralba. *Learning Deep Features for Discriminative Localization*. http://cnnlocalization.csail.mit.edu/. Published 2016.

[9] Somshubra Majumbra. keras-attention-augmented-convs Github repository. https://github.com/titu1994/keras-attention-augmented-convs. Accessed September 3rd, 2019.