Explainability of Deep Computer Vision Models for AI-Generated Faces

Nicholas Keriotis
University of Illinois Urbana-Champaign
506 S. Wright St. Urbana, IL 61801-3633
nk31@illinois.edu

Yihong Jin
University of Illinois Urbana-Champaign
506 S. Wright St. Urbana, IL 61801-3633
yihongj3@illinois.edu

Mark Bauer
University of Illinois Urbana-Champaign
506 S. Wright St. Urbana, IL 61801-3633
markb5@illinois.edu

Abstract

With the release of tools like DALLE-3, StyleGAN and SORA, computer generated imagery has become a normalized component of our daily lives. Furthermore, as generative models continue to improve in quality and increase in accessibility, generated content will only continue to grow in prevalence. While the growing capabilities of generative models are promising, they also present a danger to the trust people put in images. The phrase "seeing is believing" may no longer hold, as bad actors can easily use generative models to create fake images. These images may be used to spread misinformation, which can cast a sense of doubt on any image, real or fake. As a consequence, identifying AI generated images can be extremely valuable as a response to improvements in generation techniques. The primary goal of this project is to use principles of computer vision in combination with deep learning methods to create an explainable model capable of distinguishing real images from generated faces, and to use these findings to elucidate what features humans should look for when determining if images are AI-generated.

Video can be found at https://drive.google.com/file/d/1kAGcwyQ-xsGiFmZxWZ2JJP-Uh2DAEf1P/view?usp=sharing

1. Introduction

The objective of this project is to examine the efficacy and explainability of deep computer vision models for discriminating between real and AI-generated faces. To achieve this goal, we implemented and trained a variety of traditional CNN models and contemporary transformer models like ViT and SWIN [2], [5] using predefined architectures.

To visualize the explainability of our models, we use the popular GradCAM [6] tool which visualizes the activations of deep networks at various layers to show which parts of an input image produce the greatest response. The motivation being that since GradCAM is able to visualize activations from certain layers in a network, applying it to our models should reveal patterns in how different architectures find patterns in the training data. This in turn can help us understand what features are important to certain models and how the effects of augmentation can help or hurt models during training and inference.

2. Approach

2.1. Dataset

To achieve our goal of producing a model capable of discriminating between real and AI generated images, we first need to obtain a properly labeled dataset. We use the 140k Real and Fake Faces [7] dataset on Kaggle which contains approximately 70k examples of both real and StyleGAN [4] generated faces. The real faces were collected by NVIDIA from Flickr and the fake faces are actually a subset from a larger 1 Million Fake Faces dataset generated by StyleGAN. The dataset consists of 100k training images, 20k validation images, and 20k test images. Each image is a 256x256 RGB image with the subject's face centered in frame facing towards the camera, although we resize these to 224x224 to work with some of our transformer architectures. This dataset contains a variety of faces under a variety of conditions. Lighting, angle, presence of accessories and makeup, as well as changes in background scenery are all present. Any model wishing to succeed on this dataset needs to be able to account for the natural variety of faces and circumstances they can be found in.

Figure 1. Visualizations from the dataset. Labels of 0 indicate GAN generated faces while labels of 1 indicate real faces. Both real and fake faces can include things like makeup, glasses, and other accessories or backgrounds.

An unfortunate downside of this dataset is that the real-world photos are all quite crisp, providing clear views of the human subjects. This is slightly problematic, as many of the AI-generated images contain artifacts. While some of these artifacts are unlikely / impossible to occur in real life (ex: inconsistent shadows, incorrect anatomy), other artifacts are entirely capable of occurring in the real world (ex: blurry regions). This is problematic because many of the models trained on this dataset blindly classify all noisy images as AI-generated (a quick look at the comments on the original Kaggle post containing this dataset will make this clear). To mitigate the effects of this inconsistent training data, we employ the using of data augmentations in our training process. These augmentations are discussed in greater detail in later sections.

2.2. Model Architectures & Training

As for model architecture, we decided to examine both traditional CNN architectures as well as more modern transformer based architectures. All our models are trained from scratch, although for the more complex architectures like transformers, we utilize predefined architectures with random weight initializations.

2.2.1 CNNs

CNNs have historically been the backbone of traditional deep learning for computer vision due to their excellent abilities to extract both local and global features from images. As such, CNNs were an obvious first choice for establishing a baseline AI detection model. Our CNN architectures follow standard principals of network design - however, we also incorporate several custom blocks in order to improve our model performance and training stability.

Residual Block: Originally introduced in the ResNet [3] paper, residual blocks are a type of convolutional block that employ skip connections to propagate information forward through very deep networks. Given how deep CNNs for image classification can be, residual blocks were an obvious addition to our network. However, we do make some slight modifications to original residual block architecture described in the ResNet paper as to better suit our needs and to produce more interesting experimental results. The first modification we make is the introduction of a 1x1 convolution, used in the residual connection to modify channel counts if the number of in channels is not the same as the number of out channels (this 1x1 convolution is not used if the channel counts are equivalent). We also introduce an output normalization layer to re-normalize the model outputs, thereby accounting for any artificial 'magnitude' that the residual connection may inject into our model. Finally, we opted to substitute all ReLU activations with GELU activations. We chose to do this because GELU activations are slightly more expressive than ReLU activations when paired with normalization. Furthermore, GELU activations should also help mitigate the effects of "Dead ReLUs" (the Dead ReLU problem occurs when all non-activation layer outputs become negative, meaning that the ReLU layers prevent any information from propagating forward).

Residual Pooling: Max pooling is a commonly utilized layer in many CNN architectures (including ResNet). We replace these with our own implementation which combines this pooling operation with a residual connection. We achieve this by letting the residual connection equal a directly downsampled version of the input features, and we utilize a single Residual Block followed by standard max pooling. These two outputs are then summed and re-normalized to create a downsampling operation with a residual connection to allow for longer uninterrupted gradient flow. We utilize our custom residual blocks and residual pooling layers to produce a residual CNN called RCNN.

Double Separable Convolutions (DSCs): We implement an extension of the Depth-Separable Convolutions introduced in the Xception paper [1]. In Depth-Separable convolutions (a.k.a. group convolutions), instead of applying all filters to all channels, we apply all filters to a single channel. In the most extreme case, there is a single filter applied to each channel of the input activations, and the resulting features are concatenated together. After performing this separable convolution, we then apply a single 1x1 convolution to pass information between channels. While the original Xception paper utilized this description of Depth-Separable convolutions, we opted to extend upon this concept by also performing spatially separated convolutions. For example, we decompose a standard (3, 3) convolution into 2 convolutions, a (3, 1) followed by a (1, 3) - which while not technically equivalent, should still incur similar performance levels while simultaneously providing additional reductions to model size and improvements to speed. These blocks also contain normalized residual connections where we normalize the summed residual and learned feature maps before producing the output. We experimented with different implementations of this, mainly changing the presence of intermediate activations as done in the original Xception paper. Although not specifically examined, during preliminary tests, we found that the presence of intermediate activations hurt performance which is consistent with the findings from the original Xception paper. We used these custom downsampling and residual DSCs to create a model called SepNet which should be more efficient than RCNN.

2.2.2 Transformer Architectures

While computer vision has traditionally been dominated by CNNs, recent applications have seen transformers successfully applied to computer vision tasks like image classification and segmentation. As such, we chose to evaluate two notable image based transformer architectures - the Vision Transformer (ViT) and the SWIN Transformer.

Vision Transformers (ViT): ViT adapts the Transformer architecture for image analysis by dividing an image into fixed-size patches. These patches are then flattened, and positional embeddings are added to retain location information before being fed into a standard Transformer encoder. This approach allows ViT to leverage self-attention to process images, making it effective for tasks like classification and, as in our implementation, detecting AI-generated faces.

For our implementation, we opted to utilize a ViT backbone provided by HuggingFace. To obtain an untrained model architecture for our specific requirements, we initially cleared the pre-existing weights. Once the model was configured with fresh weights, we incorporated a custom classifier head. This new head is designed to process the feature representations output by the Vision Transformer's encoder. Our classifier head utilizes a simple feed forward network composed of linear layers, layer norms, and nonlinear activations. This sequential network ultimately reduces from the raw 1000 classes the architecture comes with and reduces it to a 2 class output.

SWIN Transformers: SWIN Transformers (short for Shifted Window Transformers) are an extension of ViT, in which a hierarchical structure is used to process images in multiple stages, gradually reducing spatial resolution while increasing feature map depth. What sets SWIN apart from ViT is its use of shifted windows, which allow for limited self-attention computation across overlapping local windows. This design reduces computational costs and enhances the model's ability to handle various scales, making it well-suited for tasks that require detailed spatial understanding.

We perform a similar resetting operation to remove any pretrained weights and modify the raw outputs of the network to conform to our two class problem setup. Since SWIN utilizes sliding windows to capture both high and low resolution features from an image, we expect this model to perform better than ViT alone.

2.3. Training Parameters

Due to the diversity of the models we examined, each model often required different optimziers or training schema. RCNN utilized the SGD optimizer with the Nesterov Accelerated Gradient with an LR of 1e-3, momentum of 0.9 and batch size of 1 due to its large size. Additionally RCNN was only trained for 15 epochs instead of the usual 25 due to its train time. Both SWIN and ViT utilized the same SGD optimizer parameters with batch sizes of 64. Finally, SepNet performed poorly with SGD, and so instead utilized the NAdam optimizer which incorporates Nesterov gradients with the Adam optimizer. SepNet used an initial LR of 1e-4 and a batch size of 32. All models were trained with a plateau scheduler for learning rate to ensure the LR was reduced whenever validation loss stopped decreasing. The models were trained for a total of 25 epochs unless otherwise noted. Additionally, we use Automatic Mixed Precision (AMP) in our training loop which casts float32 operations like the forward and backward pass to float16 to reduce the memory footprint of models.

2.4. Explainability

While we may see an image in our daily lives and wonder if it is AI-generated, we might not always have the luxury of having an AI-generated image detection model at our disposal. However, if we can learn to identify the features that these models use to classify the images, then we can improve our inherent abilities to recognize AI generated content. Additionally, explainability helps identify and mitigate biases, understand model failures, and provides insights into the decision-making processes of our models.

Grad-CAM (Gradient-weighted Class Activation Mapping) is an effective tool for enhancing model transparency. It provides visual explanations for decisions made by various types of neural networks by highlighting important regions in the input image that influenced the prediction. The visualizations produced by GradCAM consist of heatmaps superimposed over input images, where redder regions correspond to higher feature relevance (i.e. features in this region are more relevant for the given class) and bluer regions correspond to lower feature relevance. These heatmaps are produced by computing the gradients of predicted class scores from the final convolutional layers in CNNs / from the layer before the final attention block for transformers (higher gradients correspond to redder regions, lower gradients correspond to bluer regions). This capability allows users to visually verify what the model sees and considers important, thereby giving humans insight into the features they should look for when assessing if images are AI-generated.

3. Results

The evaluation of our model involved two experimental procedures: (1) assessing the accuracy of each model in detecting AI-generated faces and (2) assessing the interpretability of the GradCAM heatmaps as they pertain to identifying relevant features for AI-generated faces.

3.1. Classification Metrics

Table 1. AI-Generated Face Detection Model Accuracy on different splits. Validation graphs can be found in Figure 5

Model Name	Train Acc.	Val Acc.	Test Acc.
RCNN	**99.99%**	**91.51%**	**91.59%**
SepNet	99.96%	88.85%	89.49%
RCNN-Aug	90.26%	89.39%	89.72%
SepNet-Aug	**98.65%**	**96.74%**	**96.64%**
ViT	94.94%	85.95%	86.26%
SWIN-25 Ep	92.21%	94.52%	94.49%
SWIN-50 Ep	**97.84%**	**94.96%**	**96.22%**

Despite their relative simplicity when compared to modern architectures, we found that CNN based models performed quite well in classifying AI-generated images. This is evident from the fact that our first custom CNN architecture achieves a test accuracy of over 90% without the need for any complex design choices. However, a major problem with CNN based approaches is that they tend to overfit to the training data. This is evident from the fact that both RCNN and SepNet achieve near perfect train accuracy with subpar validation accuracies. To remedy this issue, we employ simple data augmentations that don't destroy useful visual artifacts. Data augmentation is especially important for detecting AI-generated images, as many classification models blindly classify images as being AI-generated if they contain trace amounts of noise. Examples of our injected data augmentations can be viewed in Figure 4 and visualizations can be seen in Figures 6 and 7.

Data augmentation on the largest custom CNN proved difficult, as the sheer size of the model when combined with data augmentation forced us to reduce the batch size to 1 in order to fit the model on the GPU during training (even with AMP enabled). This clearly produces worse results, as the small batch size appears to have severely hindered the optimization process. Training on larger batch sizes may prove more effective, but due to computation constraints, this was as far as we could go with RCNN. SepNet was able to achieve much greater performance, although interestingly required AMP to be disabled for effective training. While SepNet did achieve the best classification performance, it also was the most cumbersome to run, taking over 4.5 hours to train completely.

The first transformer based model we trained was ViT, which was quite poor at detecting AI-generated images. This poor performance could be improved with more fine tuning of the architecture, but given the difficulties of modifying predefined architectures and the excellent performances of CNNs, and we decided not to continue with the ViT model. The next transformer model we trained was SWIN, which was trained for both 25 and 50 epochs (since the model didn't appear to converge after only 25 epochs). SWIN was much more successful than ViT, likely due to its ability to extract features at multiple scales which can be useful for artifact detection. Additionally, it should be noted that SWIN was able to achieve its performance without the use of data augmentation, a necessity for models like SepNet.

Since we cannot find any papers that have used this dataset specifically, we compare our results to the best posted results on Kaggle from this dataset. The best results we saw were in the range of 97%-99%, although most

of these notebooks used pretrained networks (whereas we opted to train from scratch). Overall, we believe our results are competitive with other models.

Table 2. Model Comparisons with a batch size of 32

Model	# Params	Memory	Train Time	Storage
RCNN (15)	1.27 B	—	17,647s	4.95 GB
SepNet	506 M	9.5 GB	**16,189s**	1.98 GB
SWIN	**27.5 M**	**8.1 GB**	17,483s	**0.12 GB**

Comparing the best CNN and best transformer models reveals different strengths and weaknesses for both architectures. We unfortunately could not adequately profile RCNN due to its large size, although we believe much of its memory costs come from the optimizer used during training. SWIN is clearly much more efficient with its parameter count (due to the self attention mechanism and multi-scale features), although it still takes the same time to train as SepNet (likely due to the computational expense that comes with self attention). SepNet by comparison dwarfs SWIN in its parameter count, although is still competitive in terms of memory usage and train time. One advantage SWIN has over the other models is its portability - the entire model takes only a fraction of the storage space of the other models, meaning it can theoretically be deployed on devices with limited memory (like mobile applications). SWIN can be made even more efficient than the version we examined, as both RCNN and SepNet utilized a 224x224 input while the SWIN architecture we used took a 256x256 input. Therefore, using a smaller input size would lead to even more computational savings.

3.2. Explainability of Models

In order for an explainable AI-generated image detector to be useful, it must be both accurate and easily interpretable by a human - that is, a human should easily be able to identify which parts of an image that a model tend to focus on during classification, and be able to *reasonably explain* why those regions are significant. Unfortunately, it is difficult to create a quantitative benchmark for assessing these traits. Therefore, we largely rely on qualitative metrics for this part of our evaluation.

In order to evaluate the explainability of each model, we had each model predict the class labels for a series images and used GradCAM to produce heatmaps of each model's gradients during inference. The same set of images was used for the evaluation of each model. Here, we visualize activations from GradCAM on both our SWIN and SepNet, as they were our best performing models. SWIN only has a single layer norm suitable for GradCAM to visualize and results can be seen in Figure 10. SepNet provides much more interesting results, and activations for each individual layer can be visualized which provides a deeper view of the model.

3.3. Augmentation Effects on Test Data

After using GradCAM to analyze results, we found that the models don't tend to focus on obvious features in the image, so we decided to analyze how the performance of our models was effected by augmenting the test dataset. We hypothesized that some key features a model needs are color, sharp artifacts, and edge artifacts, so we implement different transforms to remove each of these and examine how models perform. This could help reveal what attributes of the image are needed for the model to function well. For the table below, we uniformly apply a single transformation to the entire test set and evaluate each model's performance. We apply grayscale transforms as well as hue shifts, Gaussian blurs of different sigma values, and center crops of 160 pixels to only show the face and not the background.

Table 3. Data Augmentation Effects on Model Performance

Model & Augmentation	Accuracy	Percent Change
RCNN - Grayscale	80.85%	-11.73%
SepNet - Grayscale	**94.56%**	**-2.15%**
ViT - Grayscale	66.30%	-23.14%
SWIN - Grayscale	84.55%	-12.13%
RCNN - Blur $\sigma = 2$	66.10%	-27.83%
SepNet - Blur $\sigma = 2$	50.26%	-47.99%
ViT - Blur $\sigma = 2$	**82.18%**	**-4.73%**
SWIN - Blur $\sigma = 2$	69.58%	-27.69%
RCNN - CC (160px)	81.75%	-10.74%
SepNet - CC (160px)	**91.27%**	**-5.56%**
ViT - CC (160px)	58.12%	-32.62%
SWIN - CC (160px)	58.83%	-38.86%

From Table 3, we see the presence of certain transforms seriously affects model performance, although different models see different performance changes. Overall SepNet appears to be the most robust both color transforms and center crops but suffers heavily from any kind of blurring, dropping to a nearly 50/50 guess even with a modest blur applied. This suggests that our models rely heavily on imperceptible features in images instead of focusing on human discernible features like misshapen facial or background features. SWIN suffers from the same blurring vulnerability, likely because it uses multi-scale features and is thus reliant on small artifacts like SepNet. Interestingly, ViT proves to be the most robust to blurring since it is forced to use low frequency features and can't easily use fine grained

artifacts to discriminate between real and fake. A full sized figure with more tests can be found in Table 4.

Actually visualizing the GradCAM activations on blurred and non blurred images shows that SepNet is clearly dependent on visual artifacts for its performance. Figures 2 and 3 show unblurred and blurred activations respectively, and we see that the unblurred activations are much stronger in uniform regions especially in maps 4 and 6 where we use column indexing to denote the map number. Additionally, before applying the blur, the model predicts the image is 98% Fake while after the blur, the model predicts the image is 99% Real instead! This helps explain both why the model performance is so sensitive to blurring and why the features aren't easily interpretable.

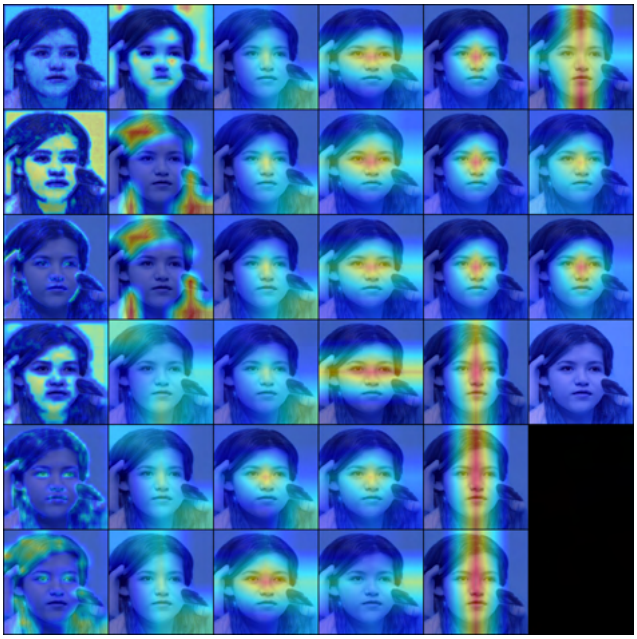


Figure 2. GradCAM visualizations of each SepNet layer with no blurring. Note the attention paid to uniform regions of the image like the face and background. This could be indicative of SepNet relying on visual artifacts which are easy to spot in uniform regions of the image. SepNet says this is Fake with 98.6% confidence.

4. Discussions and Conclusions

In this project, we have the assessed the efficacy and explainability of AI-generated image detectors. We have demonstrated that deep learning based approaches can be used to achieve solid performance in the identification of AI-generated images. More modern architectures like SWIN are very effective at identifying real and fake images, but traditional CNNs have proven just as (if not) more effective, with the caveat of their larger size. Additionally,

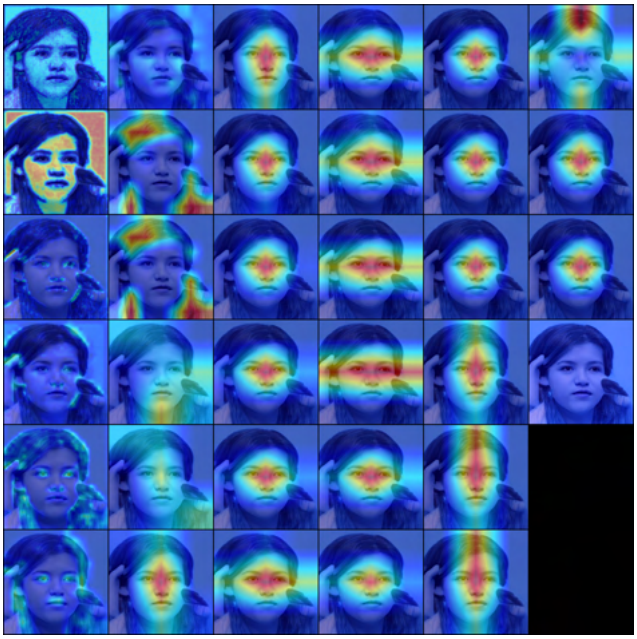


Figure 3. GradCAM visualizations of each SepNet layer when a Gaussian Blur is applied. Note the reduction of activations on layers 4 and 6. SepNet now says this is *Real* with 99.9% confidence, a complete reversal of confidence!

GradCAM activation maps are generally not easily interpretable. To ensure explainability, we likely need to force models to focus on large features in images instead of exploiting artifacts imperceptible to humans.

Future work could include finding ways to enforce this explainability, which may involve training on blurred images (which would remove artifacts and force models to find actual anatomic errors in the images). This of course could impact performance, or make training impossible especially if the GAN generated images are reasonably correct.

5. Statement of Individual Contributions

- Nicholas Keriotis: Worked on dataset infrastructure, custom CNN architectures, and training loop/model training since I have the strongest GPU of the group.
- Yihong Jin: Explored PyTorch-Grad-Cam library which is finally used to do explainable visualization. Wrote codes for SWIN Transformers which are compatible with Grad-Cam.
- Mark Bauer: Worked on integration of GradCam visualizations into original model architectures. Worked on evaluation of explainability of models.

References

[1] François Chollet. Xception: Deep learning with depthwise separable convolutions, 2017. 3

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021. 1

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 2

[4] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019. 1

[5] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows, 2021. 1

[6] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128(2):336–359, Oct. 2019. 1

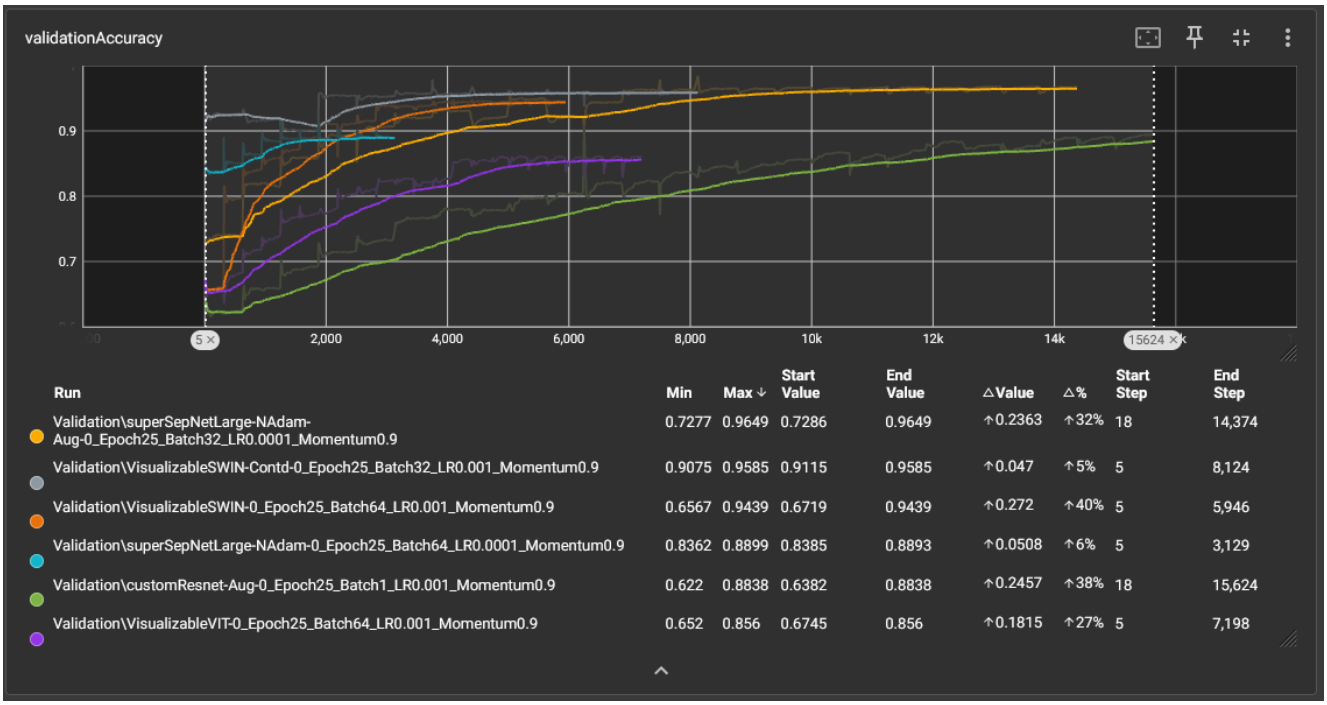
[7] Xhlulu. 140k real and fake faces, 2020. 1

Dataset can be found at https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces/data

6. Appendix

```python
TRAIN_TRANSFORM_AUG = v2.Compose([
    v2.Resize((224, 224)),  # Resize images to fit Swin Transformer input dimensions
    v2.ToImage(),
    v2.ToDtype(torch.float32, scale=True),
    v2.RandomHorizontalFlip(),
    v2.RandomResizedCrop(size=224, scale=(0.7, 1)),
    v2.RandomGrayscale(p=0.05),
    v2.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.1, hue=0.02),
])
```

Figure 4. Data augmentations used for the Custom CNN and SepNet. The augmentations are relatively simple and include very light color transforms as to not disturb the distribution of real and fake faces. This is important as if our models pick up on subtle artifacts for AI generated faces, using aggressive augmentations may introduce similar artifacts to real faces which makes discrimination much more difficult if not impossible. Additionally, we intentionally avoid Gaussian Blurring as we hypothesize that many of our models require detection of very high frequency features to make proper predictions.



Run	Min	Max	Start Value	End Value	△Value	△%	Start Step	End Step
Validation\superSepNetLarge-NAdam-Aug-0_Epoch25_Batch32_LR0.0001_Momentum0.9	0.7277	0.9649	0.7286	0.9649	↑0.2363	↑32%	18	14,374
Validation\VisualizableSWIN-Contd-0_Epoch25_Batch32_LR0.001_Momentum0.9	0.9075	0.9585	0.9115	0.9585	↑0.047	↑5%	5	8,124
Validation\VisualizableSWIN-0_Epoch25_Batch64_LR0.001_Momentum0.9	0.6567	0.9439	0.6719	0.9439	↑0.272	↑40%	5	5,946
Validation\superSepNetLarge-NAdam-0_Epoch25_Batch64_LR0.0001_Momentum0.9	0.8362	0.8899	0.8385	0.8893	↑0.0508	↑6%	5	3,129
Validation\customResnet-Aug-0_Epoch25_Batch1_LR0.001_Momentum0.9	0.622	0.8838	0.6382	0.8838	↑0.2457	↑38%	18	15,624
Validation\VisualizableVIT-0_Epoch25_Batch64_LR0.001_Momentum0.9	0.652	0.856	0.6745	0.856	↑0.1815	↑27%	5	7,198

Figure 5. Validation accuracies for some of the models examined. If the learning rate from the LR scheduler dropped below 1e-7, we considered training to be finished and terminated the training loop, explaining the sudden stops in some models.

Figure 6. Real augmented faces

Figure 7. Fake augmented faces

```python
class DoubleResidualDWSeparableConv2d(nn.Module):

    """
    Defines a residual double separable convolution
    """

    def __init__(self, in_channels, out_channels, activation:nn.Module=nn.GELU(), kernel_size:int=3):
        super(DoubleResidualDWSeparableConv2d, self).__init__()

        self.activation = activation

        self.conv1 = DoubleDepthwiseSeparableConv2d(in_channels=in_channels, out_channels=out_channels, activation=activation, kernel_size=kernel_size, padding='same')
        self.conv2 = DoubleDepthwiseSeparableConv2d(in_channels=out_channels, out_channels=out_channels, activation=activation, kernel_size=kernel_size, padding='same')

        self.outNorm = nn.BatchNorm2d(num_features=out_channels)

        # If in channels != out channels, pass through a 1x1 convolution to change channel count. Sizes must be the same still!
        if in_channels != out_channels:
            self.residualLayer = nn.Sequential(
                nn.Conv2d(in_channels=in_channels, out_channels=out_channels, kernel_size=1, bias=False),
                nn.BatchNorm2d(num_features=out_channels)
            )
        else:
            self.residualLayer = nn.Identity()

    def forward(self, x):

        residual = self.residualLayer(x)

        y1 = self.conv1(x)
        y2 = self.conv2(y1)

        # Do the residual connection by adding inputs to outputs
        return self.outNorm(y2+residual)
```

Figure 8. The residual DSC block which uses the block in Figure 9 as a building block.

Table 4. Data Augmentation Effects on Model Performance

Model & Augmentation	Accuracy	Percent Change
RCNN - Grayscale	80.85%	-11.73%
SepNet - Grayscale	**94.56%**	**-2.15%**
ViT - Grayscale	66.30%	-23.14%
SWIN - Grayscale	84.55%	-12.13%
RCNN - Hue Shift	68.46%	-25.25%
SepNet - Hue Shift	**91.31%**	**-5.52%**
ViT - Hue Shift	67.14%	-22.17%
SWIN - Hue Shift	76.41%	-20.59%
RCNN - Blur $\sigma = 1$	**88.31%**	-3.58%
SepNet - Blur $\sigma = 1$	85.12%	-11.92%
ViT - Blur $\sigma = 1$	84.68%	**-1.83%**
SWIN - Blur $\sigma = 1$	81.77%	-15.02%
RCNN - Blur $\sigma = 2$	66.10%	-27.83%
SepNet - Blur $\sigma = 2$	50.26%	-47.99%
ViT - Blur $\sigma = 2$	**82.18%**	**-4.73%**
SWIN - Blur $\sigma = 2$	69.58%	-27.69%
RCNN - CC (160px)	81.75%	-10.74%
SepNet - CC (160px)	**91.27%**	**-5.56%**
ViT - CC (160px)	58.12%	-32.62%
SWIN - CC (160px)	58.83%	-38.86%

```python
class DoubleDepthwiseSeparableConv2d(nn.Module):

    """
    Implements separable depthwise-pointwise convolution for faster performance. I think this was used in EfficientNet
    https://www.youtube.com/watch?v=vVaRhZXovbw&list=WL&index=44

    This Double separable version separates into both vertical and horizontal convolutions for max efficiency
    """

    def __init__(self, in_channels, out_channels, activation:nn.Module=nn.GELU(),
                 kernel_size:int=3, stride:int=1, padding:int='same'):
        super(DoubleDepthwiseSeparableConv2d, self).__init__()

        self.activation = activation

        if stride != 1:
            padding = 0

        # Spatial and channelwise separable convolution should improve speed while hopefully keeping performance the same
        # No activation because original paper says that's better
        self.depthwise = nn.Sequential(
            nn.Conv2d(in_channels=in_channels, out_channels=in_channels,
                      kernel_size=(kernel_size, 1), stride=stride, padding=padding, groups=in_channels, bias=False),
            # self.activation, # <== Having these hurts performance! Just like the paper said!
            nn.Conv2d(in_channels=in_channels, out_channels=out_channels,
                      kernel_size=(1, kernel_size), stride=stride, padding=padding, groups=in_channels, bias=False),
            nn.BatchNorm2d(num_features=out_channels),
            # self.activation, # <== Having these hurts performance! Just like the paper said!
        )

        self.pointwise = nn.Sequential(
            nn.Conv2d(in_channels=out_channels, out_channels=out_channels, kernel_size=1, stride=1, padding=0, bias=False),
            nn.BatchNorm2d(num_features=out_channels),
            self.activation,
        )

    def forward(self, x):

        y1 = self.depthwise(x)
        return self.pointwise(y1)
```

Figure 9. A single Double Depthwise Separable Convolution block. This is the component used in the residual version.

Figure 10. SWIN GradCAM Visualizations. SWIN only has a single layer suitable for visualization, but we see the model focuses primarily on the background while less attention is paid to features like the eyebrows and mouth.

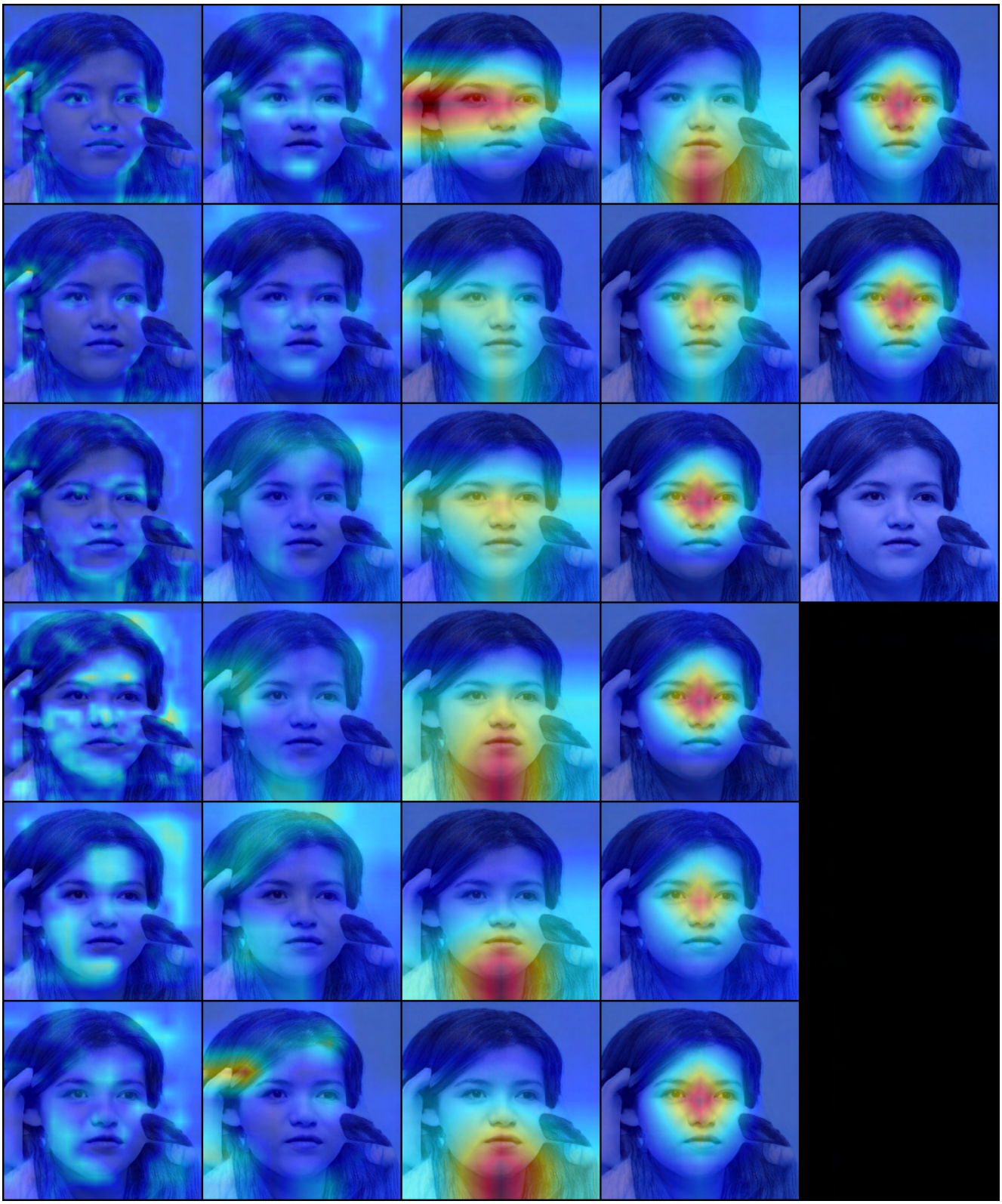


Figure 11. GradCAM activations from RCNN. This model actually incorrectly classifies the image with confidences of 88% Real and 12% Fake. The activations appear to be more scattered and only become more coherent in deeper layers where the receptive field is much larger.

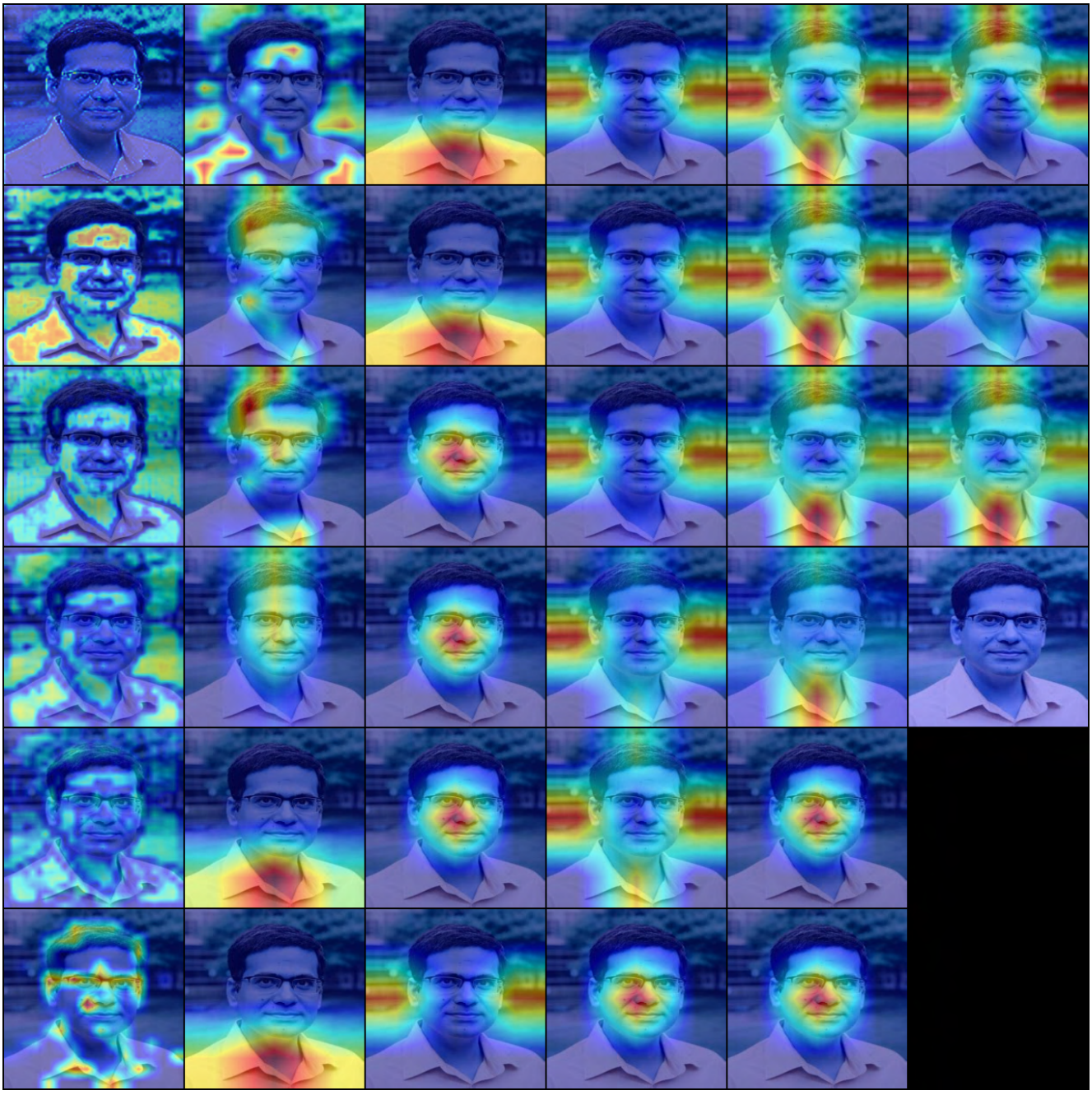


Figure 12. SepNet activations on the professor. Interestingly, SepNet says the image is Fake with 80.23% confidence, although this may be a side effect of the original image not being the standard 256x256 image like the other images in the dataset were.