Augmenting the Fashion-MNIST Dataset with an Auxiliary Classifier GAN

Kush Attal, Jay Choi, Abhi Jha, Fady Gouda
Washington and Lee University
Lexington, Virginia
attalk21@mail.wlu.edu

ABSTRACT

Image-based classification has long been a focus of deep learning. A variety of approaches have been developed with the goal of perfecting this process, with some of the most effective being the use of a Convolutional Neural Network (CNN) or Generative Adversarial Network (GAN). With that in mind, the goal of our work is to use these two model frameworks in conjunction to create a GAN that will be able to accurately generate images from data. In approaching this task, we choose to implement an Auxiliary Classifier GAN (AC-GAN) because it is deemed the most appropriate model design for the challenge at hand. As an extension of the conditional GAN (cGAN), the AC-GAN is unique in that it changes the discriminator to independently predict the class label of a given image rather than receiving it as input. In the context of our work, the CNN serves as an external "evaluator" of sorts for synthetic data generated by the AC-GAN, which will ideally serve to augment the existing dataset used by the CNN for training. Once training is complete, the AC-GAN should be able to effectively replicate input data such that the CNN will be able to classify the generated data with high performance. While the final performance of the CNN on the generated dataset was suboptimal, the process of creating this AC-GAN exposes plenty of potential opportunities for data augmentation.

CCS CONCEPTS

• **Deep Learning**;

KEYWORDS

AC-GAN, Deep Learning,

ACM Reference format:
Kush Attal, Jay Choi, Abhi Jha, Fady Gouda. 2021. Augmenting the Fashion-MNIST Dataset with an Auxiliary Classifier GAN. In *Proceedings of Final Project, Lexington, VA, April 2021 (AI '21),* 11 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Image-based classification has long been a focus of the deep learning. Consequently, a variety of approaches have been developed with the goal of perfecting this process, with some of the most effective being the use of a Convolutional Neural Network (CNN) or Generative Adversarial Network (GAN). With that in mind, the goal of our work is to use these two model frameworks in conjunction to create a GAN that will be able to accurately generate images from data. In approaching this task, we choose to implement an Auxiliary Classifier GAN (AC-GAN) because it is deemed the most appropriate model design for the challenge at hand. As an extension of the conditional GAN (cGAN), the AC-GAN is unique in that it changes the discriminator to independently predict the class label of a given image rather than receiving it as input. This allows the training process to be stabilized, and also allows large high-quality images to be generated whilst the model learns a representation in the latent space that is independent of the class label [6]. In the context of our work, the CNN serves as an external "evaluator" of sorts for synthetic data generated by the AC-GAN, which will ideally serve to augment the existing dataset used by the CNN for training. Once training is complete, the AC-GAN should be able to effectively replicate input data such that the CNN will be able to classify the generated data just as well as it classifies the actual data. In terms of the data itself, our models will be using the publicly available Fashion-MNIST dataset from an arXiv paper by Xiao et. al, 2017. The dataset contains a number of grayscale images depicting articles of clothing that fall under 10 different fashion categories [14] [16]. Additional details regarding the data will be presented in the Data Extraction section of this paper.

2 RELATED WORKS

As previously mentioned, the use of image-based GANs is fairly well-explored. One field of study in which image-based GANs have become particularly useful is medicine, where the ability to generate accurate synthetic data is vital for a few reasons. Firstly, while supervised deep learning is already state of the art in many medical image analysis tasks, the success of these models depends heavily on the availability of labeled training data. However, this is often a major source of concern because such data is quite scarce due to the costly nature of medical image acquisition and labeling. Effective image-based GANs can alleviate this data shortage in a much less time-consuming manner than manual processing/labeling of medical images, which makes them extremely valuable. In addition, another problem faced by supervised deep learning models used in medicine is class imbalance given that many pathologies are extremely rare. Again, a proper image-based GAN can be vital to resolving this issue by accurately replicating data on rare pathologies such that supervised deep learning models can be properly trained to recognize these conditions [11]. The use of image-based GANs has also been applied to a variety of other problems, with one such application being somewhat similar to our work: clothing

AI '21, April 2021, Lexington, VA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

design. The use of image-based GANs in fashion makes it possible to inherit design ideas without a fashion designer and gain general inspiration. One paper by N. Kato et. al, 2019 utilizes a Progressive Growing of GANs (P—GANs) approach to generate a pattern maker that aids in clothing design [10].

There also exists related papers that use CNNs and GANs in combination as our work does. In these cases the objective of using these two model architectures together is universal; the GAN serves to augment the data available to train the CNN. One such example of this approach is that used by R. Gandhi et. al, 2018, where the goal is to complete image-based classification for plant disease detection. In this case, GANs are used to augment the limited number of local images of diseased plants that are available in India [8]. Similarly, in a paper published by D. Vint et. al, 2021, the authors use CNNs and GANs in conjunction to solve the problem of automatic target recognition in Low Resolution Foliage Penetrating Synthetic Aperture Radar images. In this context, GANs are again used to augment the training set available for the CNNs, which serve to extract both low and high-level features of the imaged targets [15].

3 DATA PREPROCESSING

3.1 Data Extraction

Our first element of learning - training data - contains all information for training our models, including the dataset of images X (with Xtrain) and the target variable y (with ytrain). In this case, it comes from the well-known and well-documented Fashion-MNIST (Modified National Institute of Standards and Technology) dataset from an arXiv paper by Xiao et. al, 2017. The dataset itself is a 10-class classification dataset extracted via Keras with 60,000 28x28-pixel grayscale images of the 10 fashion categories as well as an additional test/evaluation set of 10,000 images, each with object labels that we wish to learn about [14] [16]. Since the images are grayscale, they will range from values of 0 - pixel value of black - to 255 - pixel value of white.

- **Initial Exploratory Data Analysis**

While initial exploration led us to reverse the pixel values (e.g., change 0 to 255 and vice versa) in order to view the clothing items in black, training the models does not require us to reverse the pixel values for data preprocessing. Initial data analysis also showed the first 10-100 clothing items of the dataset, ranging from shoes to T-shirts to ankle boots. Some basic stats on the features include: grayscale square images, numerical digit classifications, and clear edges between the black items and white background. These visual, basic features like edges and grayscale gradients are essential for the CNN to filter out in order to build more complex, hierarchical features for classification in the later layers.

- **Input**

Set of data fed into the model, such as the preprocessed subset Xtrain from the gathered Data, which includes 60,000 colored and normalized 2D vectors representing 60,000 handwritten-single-digit images from the Fashion-MNIST dataset [14]. Both datasets are used to train all of the models used in this project

- **Output**

Result generated from the model to be compared and analyzed, in this case to ytrain or the classification of each of the 60,000 images as one of 10 categorical class labels represented by a numerical digit. These categories include:

(1) Trouser
(2) Pullover
(3) Dress
(4) Coat
(5) Sandal
(6) Shirt
(7) Sneaker
(8) Bag
(9) Ankle Boot
(10) (0) T-shirt/top [14].

After creating the learning models, the output also includes the result generated from the model to be compared and analyzed, in this case to ytrain. Both models/design architectures - the CNN and the Auxiliary Classifier GAN (AC-GAN) - will be trained with the same dataset.

3.2 Preprocessing

After importing the necessary library imports (pyplot, keras) and CIFAR dataset (already grayscale images in multidimensional Numpy arrays) via load-data() into Xtrain, ytrain, Xtest, ytest, further preprocessing was necessary for each of the models.

Regarding the preprocessing for the CNN, the data was loaded and reshaped into a single channel since the colors were only grayscale. Later, the labels of the dataset were one-hot encoded. With one-hot encoding, these class integer labels are converted into 10-element binary vectors with 1's at the index position of the class value. However, the dataset still had to be preprocessed prior to fitting with a CNN (which accepts numerical inputs). This meant normalizing the Xtrain and Xtest datasets from values of [0,255.] (pixel values) to [0, 1] via division with 255. In addition, training and testing data were converted to float values. After further preprocessing and normalizing (mentioned above) to not have to deal with a wide array of input values and not have the weights range so drastically to better aid with preventing large gradients for certain weights overshadowing smaller gradients for other weights, the initial model (hyper parameters mentioned above) is developed.

Regarding the preprocessing for the AC-GAN, the dataset was loaded similarly into Numpy arrays; however, since the images are grayscale in 28x28 arrays, an additional channel was needed for a three-dimensional input for the AC-GAN. In addition, the datasets were normalized from [0, 255.] to [-1, 1] since the generator portion of the AC-GAN will output generated images in this range. However, this scaling functions for the similar purpose of preventing a huge range of values for the pixels.

4 APPROACH

4.1 Model Implementation Methodology

After preprocessing the dataset, the models were developed according to the 4 remaining elements of learning:

(1) **Target Function** - The unknown function that accurately maps all input values to output values. In the case of the CNN, the target function maps grayscale, normalized images to the classification of t-shirt, trouser, boot, etc. In the case

of the generator for the AC-GAN, the target function maps class labels with a noise vector to generated images similar to those of the Fashion-MNIST datasetIn the case of the discriminator model of the AC-GAN, the target function maps the input image to (1) the classification of t-shirt, trouser, boot, etc. and (2) the probability that the image is real or fake

(2) **Final Hypothesis** - the learning algorithm's best estimate of the target function that maps input values to output values given the hypothesis set (see below). In this case, with the task of multiclass classification for the CNN, the final hypothesis takes input values of the preprocessed images and, using the best performance of the hypotheses, outputs target labels using a combination of linear functions (such as dot product of transposed weight and input matrices) and nonlinear functions (such as ReLU in convolutional layers and softmax function) that is limited by hyper parameters such as the number of hidden layers, nodes, learning rate, and epochs. In the case of the generator for the AC-GAN, the final hypothesis takes input values of the label and random noise vector and outputs generated images with a similar combination of linear and nonlinear functions. In the case of the discriminator for the AC-GAN, the final hypothesis takes input values of the images and outputs classifications and likelihood that the image is real or not.

(3) **Hypothesis Set** - set of all possible hypotheses/candidate formulas that may map input values to output values, which is limited by the model and the learning algorithm (see below) to an extent. With a CNN, the forward pass equations that may map inputs to outputs is a combination of linear functions (dot product of transposed weight matrices * input vector or filter * input vectors) fed into nonlinear functions (such as ReLU) fed into more linear functions and later nonlinear functions (depending on the number of hidden layers) eventually fed into an softmax layer to create the output values for classification.With the generator of an AC-GAN, the process is similar to that of a reverse CNN, taking down-sampled noise vectors and up-sampling them (with the label) into complete images. This process is accomplished with additional features such as transpose convolution - which allows developers to upsample features from the input into a larger convolved features (similar to deconvolution) - and additional linear and nonlinear functions that will be explained later. With the discriminator of the AC-GAN, the process is similar to that of a standard CNN, with the additional dense layer used for outputting whether the image is fake or real (similar to binary classification).

(4) **Learning Algorithm** - method(s) used to tune the model and its weights and eventually prune the hypothesis set to find the best/final hypothesis. In this case, to create a classification CNN, the learning algorithm's "feedback signal" that it uses to tune the weights lies partially in the use of back-propagation to calculate the gradients (partial derivates of the loss function with respect to the specific weight) and gradient descent to adjust the model's parameters to minimize the categorical cross entropy loss function.

With the AC-GAN, especially the discriminator, the model uses 2 loss functions - (1) the binary cross entropy loss function for binary classification of real or fake and (2) the categorical cross-entropy loss function for class label classification. More specifically, the AC-GAN follows a similar approach to minimax GAN loss; according to a GAN research article, while "the discriminator seeks to maximize the maximize the mean of the log probability of real images and the log of the inverse probability for fake images...the generator seeks to minimize the log of the inverse probability predicted by the discriminator for fake images" [11]. In other words, with AC-GAN, the discriminator is working to learn the difference between real and fake images and maximize this difference while the generator seeks to minimize that difference; however, while the two components are adversarial, they also "teach each other" to improve. Once the generator is able to create images that the discriminator is 50/50 probable whether it is real or fake, the generator has reached an adequate level of image generation. This learning algorithm also uses gradient descent to adjust the design architecture's parameters.

4.2 General Model Structure

After preprocessing, the initial CNN model was created with the specified requirements:

- 6 convolutional layers with ReLU, HeUniform weight initialization, same padding and 3x3 filters ranging from size of 32 to 128,
- 3 max pooling layers with 2x2 filters,
- Batch Normalization after each convolutional layer and Dropout of 0.2 after each max pooling layer,
- 1 128-sized-Dense/fully-connected (FC) layer with ReLU activation and HeUniform with subsequent BatchNormalization and Dropout
- 1 Softmax layer of size 10,
- and Adam optimizer with categorical cross entropy loss and accuracy metrics.

The initial AC-GAN model was created with the specified requirements: For the discriminator,

- 4 convolutional layers with Leaky ReLU, Guassian weight initialization, same padding, and 3x3 filters, some with 2x2 strides, ranging from size of 32 to 256
- Batch Normalization after each of the final 3 convolutional layer and Dropout of 0.5 after each convolutional layer
- 2 Dense/FC layers
  - 1 with sigmoid activation for binary classification
  - 1 with softmax activation for categorical classification
- and Adam optimizer with binary and categorical cross entropy loss functions

For the generator,

- an Embedding layer FC layer to interpret the classification label as another feature map for the generator
- an additional FC layer with ReLU activation to combine the feature map for the label with the feature maps for the output image
- 1 Transpose Convolutional layers with ReLU, Guassian weight initialization, same padding, and 5x5 filters with 2x2 strides

to upsample the features maps from size 7x7 to 14x14 in preparation for the output image
- Batch Normalization after the initial convolutional layer
- 1 Transpose Convolutional layers with tanh activation, Guassian weight initialization, same padding, and 5x5 filters with 2x2 strides to upsample the features maps from size 14x14 to 28x28 in preparation for the output image

After model creation and initial fit testing, each model was later fit to the dataset of the 60,000 training images via 200 and 100 epochs for the CNN and AC-GAN, respectively. Batch sizes of 64 are used, indicating that 937 images are used to train the models per epoch. For the CNN, this involves training on batches of the original Fashion-MNIST dataset. For the AC-GAN, this involves training the discriminator first on a batch of real images and then fake images.

After training, the CNN will be evaluated on the test set of the original Fashion-MNIST dataset. The AC-GAN will be used to generate additional images based on the Fashion-MNIST dataset that will be evaluated by the CNN for generalizability.

For additional detail and rationale of each step, the code can be divided into the following tasks:

4.3 Developing the CNN

After data preprocessing, the CNN is developed. The model architecture, popular for image classification, is a supervised learning algorithm and - as a deep learning model - can learn non-linear functions for classification. The initial layer is a convolutional layer of 3x3 filters with size 32. These filters aid with minor dimensionality reduction and also parsing out specific features in images represented irregardless of variations in size, shape, color, illumination, etc. [2].

The layer also has ReLU activation function, substituting negative pixel values with 0 and treat positive pixel values with the identity function. This pixel-by-pixel operation is useful to only parse out the positive features used for a simple representation of the dataset. This simple representation may be fed into the dense/FC layer that can use weights and traditional matrix multiplication for classification. For this CNN, ReLU is useful for selecting complex hierarchical features that either contribute to classification or do not contribute at all since it is a piecewise activation function between 0 (for negative values) and the identity function (for positive values). Unlike the sigmoid logistic activation function, ReLU has a reduced likelihood for vanishing gradients since the derivative of the function is either 0 (when negative) or 1 (when positive) [13]. However, since we cannot solve for the gradient at 0 with ReLU, initializing values to 0 would lead to an error with the network, a disadvantage to be weighed with ReLU variants commonly used for convolutional layers.

The layer also uses HeUniform kernel for initial values for the kernel/filter values. According to the Keras default implementation, the values are initialized via a variant of the Glorot uniform distribution. Values are drawn from a normal distribution within the limits of sqrt(6/incoming links for the layer). This initialization is to avoid large initial weights that may generate large signals in the activation function lead to vanishing/exploding gradients at the beginning of training [3]. Additionally, same padding is used which changes padding around an image so that the filter and stride match, leading to a balanced output. Compared to full padding which considers all values as equal, same padding does not increase the output's dimensionality to the same extent, improving feature identification. Unlike valid padding which drops information outside of the stride and filter size, same padding maintains all information in the input for downstream analysis, which may improve performance.

In addition, this first convolutional layer is unique in that it determines the size of the expected input array which is a grayscaled 28x28 image. After this convolutional layer is a batch normalization layer. Batch normalization can help reduce the issue of vanishing/exploding gradients during training. The process itself is just zero-centering and normalizing each input followed by scaling and shifting of the result with 2 additional parameters. An advantage is the model can learn the best scale and mean for each layer's input, possibly replacing the standardization of input features and even acting as a regularization technique to supplement other techniques like Dropout. This technique may also speed up training by allowing higher learning rates [9].

After 2 sets of convolutional and batch normalization layers, a max pooling layer with a 2x2 filter is introduced to extract the highest-valued features from the input and further reduce the dimensionality of the input. Max-pooling as a technique helps block out noise and pick the most prominent feature to use as a complex hierarchical feature for future layers to build on and as a representation of the dataset for downstream classification. While it may account for outliers and not consider other important, secondary features like mean-pooling, it is more computationally efficient which is useful for our classification of 60,000 images for training.

After 3 sets of convolutional and max pooling layers with batch normalization and dropout, the extracted features is fed as an input into a 128-sized dense or fully connected layer with ReLU activation function and HeUniform weight initialization. As mentioned prior, the fully connected (FC) layer is useful as a layer that can use weights and traditional matrix multiplication for classification, similar to our feed-forward neural networks. With the supplementation of the convolutional and pooling layers, the FC layer can use the extracted features for improved classification performance than our solitary FFNN or MLP Classifiers could achieve with color images. As mentioned before, RelU is useful (relative to sigmoid activation functions) for combatting vanishing gradients, and HeUniform weight initialization is useful for preventing vanishing or exploding gradients.

Afterwards, like our basic feed-forward neural networks, the FC layer's transformed signal is propagated (with a final set of weights) to each of the 10 output nodes of the output layer, following the well-known pattern of summation and transformation. While each output node sends a logit (un-normalized value) representing the likelihood that the input image is a certain classification of the target variable, the logits are fed into a softmax layer, in which each of the 10 Softmax units output a number associated with the percentage of likelihood that the input example belongs to a certain class. These units are governed by a winner-takes-all approach, leading a single classification to be 1 and the rest to be 0 to help drive the error. This softmax layer is especially useful to classification with mutually exclusive labels of the target variable. These predictions are then used to tune the weights with the categorical cross-entropy loss

function, a popular loss function with a softmax layer where a number of different classes fit the input values. The loss function itself is similar to the log-loss or negative log-likelihood function used for Logistic Regression; however, cross entropy is useful when using the softmax layer and multiple classes to incorporate a similar form of log-loss for all of the output nodes. The error vector is calculated for the incorrectness that model generates, and it can use the error to change either the features are extracted during the convolutional layers or the traditional gradient descent to adjust weights for the dense layers.

The entire model also uses the Adam optimizer which is a popular tool used to calculate the adaptive learning rates for each parameter [4]. Regarding efficiency, having an ideal adaptive learning rate may be more efficient (take less time to reach the same performance) than a constant learning rate. Relative to other learning rate algorithms like Adagrad, Adadelta, and RMSProp, Adam combines momentum and RMSProp to (1) stabilizing against exploding and vanishing gradients since the update, like RMSProp, has a "normalizing/scaling" step, (2) create a step size derived from moving or running average of previous gradients, (3) increase efficiency when training a model that can converge quickly and (4) include bias correction that can improve the speed of convergence.

4.4 Developing the AC-GAN

Note that the model architecture of the AC-GAN used in our work is heavily influenced by that provided by J. Brownlee on Machine Learning Mastery in How to Develop an Auxiliary Classifier GAN (AC-GAN) From Scratch with Keras (2019) [6].

- **Discriminator**

Many of the components seen in the design of the discriminator are similar to that of the CNN and thus do not deviate from their prior descriptions. However, the discriminator does have some unique components.

For example, leaky ReLU is used instead of standard ReLU, which allows a small nonzero slope for negative values. One of the major benefits for leaky ReLU vs standard ReLU is speeding up training, which is beneficial when training GANs that may take several hours to develop [12]. In addition, Guassian distribution is used for value initialization; however, this is still a normal distribution, just with a pre-defined standard deviation to speed up training and should not deviate too sharply from HeUniform value initialization. Additionally, a 2x2 stride is introduced, replacing pooling layers for the function of downsampling the input image since the computation is more efficient and even improves model accuracy in some cases with GANs [1]. Ultimately, many of the changes used for the discriminator were used to increase the speed of training.

Another major addition is the use of a binary cross-entropy loss function or log-loss function to determine if an image is real or fake. In this case of only 2 possible classes (binary classification), the log-loss is used as the loss function, and as a result, we have subjected our weights to the sigmoid function which squashes values between 0 and 1. This operation is useful for probability analysis between two mutually exclusive classification (i.e., real or fake). With gradient descent, this loss function is altered according to the function of the discriminator: determining and maximizing the difference between real and fake images (as mentioned above). However, this minimaxing is performed concurrently with training the generator portion of the AC-GAN, which will be discussed below.

- **Generator**

As previously mentioned, many of the components for the design of the generator are similar to the CNN and discriminator from above. However the generator has some unique components.

Namely, the generator model begins with an Embedding layer followed by a Dense layer. This function is an effective approach to convert the class label into a feature map that can be added to the other feature maps used early in training the generator [6]. In other words, an embedding with a nominal number of dimensions is used with a FC layer to develop a 7x7 feature map representing the class label. As a result, the subsequent Dense layer can add this 7x7 feature map to the other feature maps used to create the output image. This function is useful in conditional GANs like AC-GAN to allow conditional generation of images, or creation of images based on a preferred class label [5]. In other words, with these Embedding and Dense layers, images of T-shirts specifically can be generated, as an example.

Additionally, convolutional transpose layers are used to upsample the images from 7x7 arrays to 28x28 arrays necessary for the output image. As previously mentioned, these layers can be seen as the reverse process for a CNN, taking an initially small noise vector and transforming it into a larger output image. This function is useful to transform an array into a larger array while maintaining the same general pattern and features present in the original array [7]. The initial convolutional transpose layer is followed by ReLU instead of leaky ReLU for the benefit of only prioritizing positive values and disregarding negative values; however, that does come at the price of losing some information about the original input image. Additionally, the final convolutional transpose layer is followed by a tanh activation function in order to match the normalization of the images performed in the data preprocessing step (converting pixel values of [0, 255] to [-1, 1]). Tanh has the advantage that negative inputs will be strongly negative (close to -1) and positive inputs will be strongly positive (close to 1). Since it is centered at 0 with its maximum derivative at 0, the most meaningful change for gradients will be at around 0, unlike ReLU. Some downsides with tanh is that there are large swings in weight values and lead to overcompensation, so the tanh function is commonly used for binary classification as is the case with this AC-GAN. The overall purpose of the generator is to create fake images indistinguishable from real images, a binary task. The tanh is also useful to overcome the vanishing gradient problem that other activation functions (i.e., the sigmoid function) is susceptible to.

After developing these models separately, the AC-GAN generates images that are tested on the trained CNN in order to determine if the Fashion-MNIST dataset can be augmented with AC-GAN generated images that a trained CNN can evaluate.

5 EVALUATION

5.1 CNN on FASHION-MNIST Training Data

Training our CNN model on the FASHION-MNIST training set results in ideal results being achieved fairly quickly. Within 10 epochs, accuracy has exceeded 94 percent and loss has decreased to

a value around 0.15. With early stop training implemented, our CNN halts training at epoch 13 with a final accuracy of 95.72 percent, a final loss of 0.1152, a final validation accuracy of 93.35 percent, and a final validation loss of 0.2019. When the CNN is run on the testing data, performance is also strong, with a final accuracy of 93.35 percent and a final loss of 0.2019 on the last epoch (313). As seen in the Figure 1 depicting the metric for accuracy, both accuracy and loss show consistent upward and downward slopes, respectively, over time, which is ideal.

5.2 Accuracy of AC-GAN discriminator

The main goal of the AC-GAN is to generate images that fall within a certain class. From a subjective visual standpoint, the AC-GAN was successful in doing so, generating hundreds of images that seemed to match their desired class. Examples of these generated images can be seen in Figure 2-7, depicting a range of fashion items such as ankle boots, sneakers, and dresses.

5.3 CNN on GAN-generated images

Unfortunately, the accuracy of the trained CNN learning model on the GAN-generated dataset was only 11 percent. Therefore, it is likely that our CNN model is likely guessing on the generated dataset. It is possible that the AC-GAN dataset is substantially different from the Fashion-MNIST dataset, the AC-GAN was not trained properly or long enough despite the images looking decent, or the method of importing the images from the AC-GAN dataset and preprocessing them for the CNN model was flawed. However, it is possible that additional epochs of training for the AC-GAN would develop images that are more similar to the Fashion-MNIST dataset.

6 THREATS TO VALIDITY

Threats to Validity of our work are mainly centered around failed training of our AC-GAN and/or overfitting. For example, since our AC-GAN model failed to generate adequate images for CNN evaluation based on the FASHION-MNIST data it was trained on, our model architecture would be require additional processing and development. In addition, it is possible that our CNN and AC-GAN models exhibit some level of overfitting to the data. Within the context of our work alone this may lead to better results in terms of accuracy/loss, but the presence of overfitting will diminish the generalizability of our model architecture, which would discredit the overall success of our work.

7 CONCLUSIONS AND FUTURE WORK

There is undoubtedly more work to be done on perfecting the ability to classify image-based data by using CNNs and GANs. With that in mind, however, we believe that our model architecture can add another piece to the puzzle. In using a CNN and AC-GAN in conjunction on the FASHION-MNIST dataset, we have created two different models that serve two very useful purposes. The CNN architecture we use should be widely applicable to a variety of image-based classification tasks, and the AC-GAN architecture, with additional proper tranining, should be generalizable to fulfill a variety of data augmentation needs. The nature of these two models and their usefulness in addressing a variety of issues makes them quite useful universally, as long as they are built correctly, effective based on evaluation, and adequately generalizable.

h

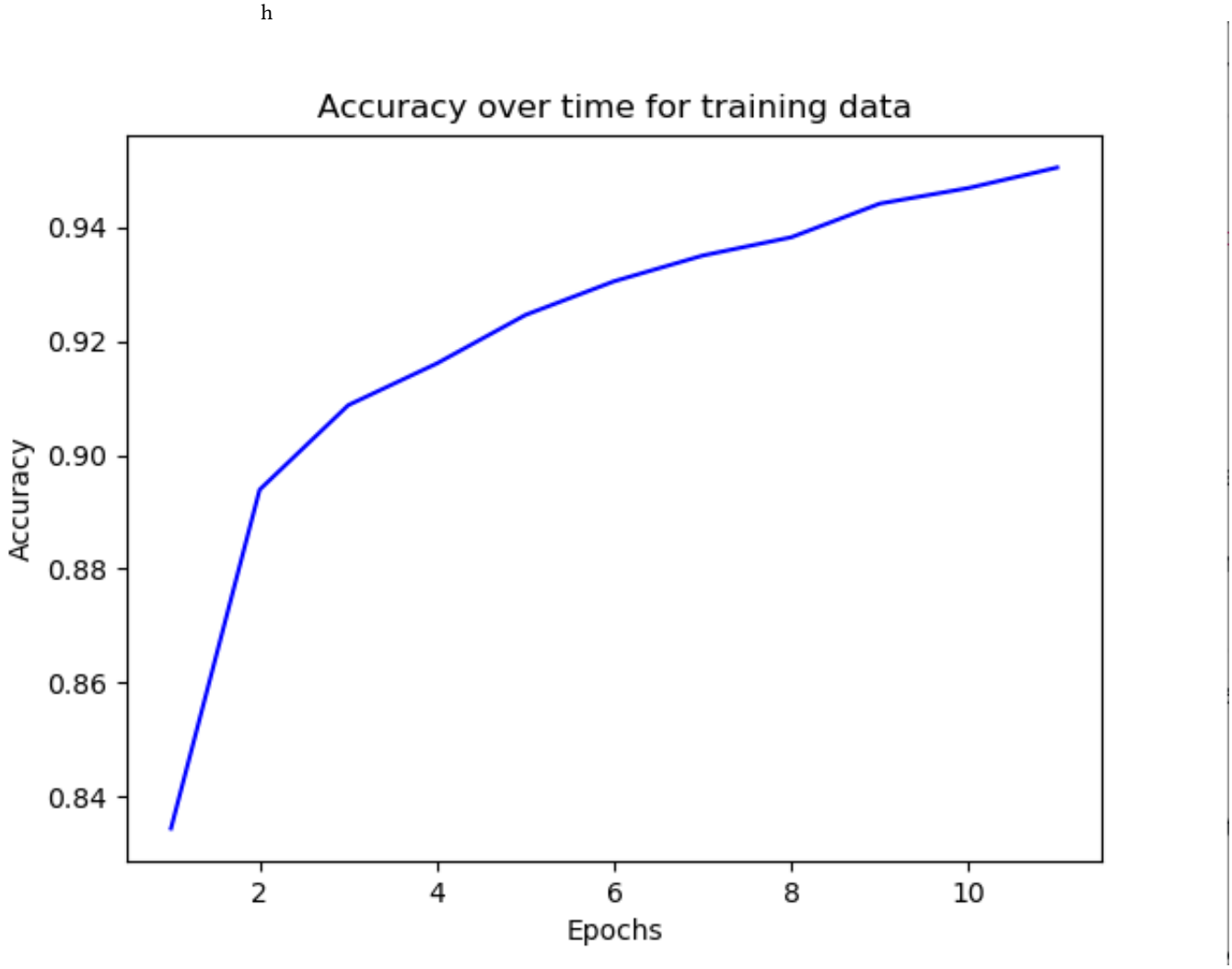

Figure 1: Accuracy of CNN Overtime on Fashion-MNIST Training Dataset

h

Figure 2: Image of T-shirt generated from AC-GAN

h

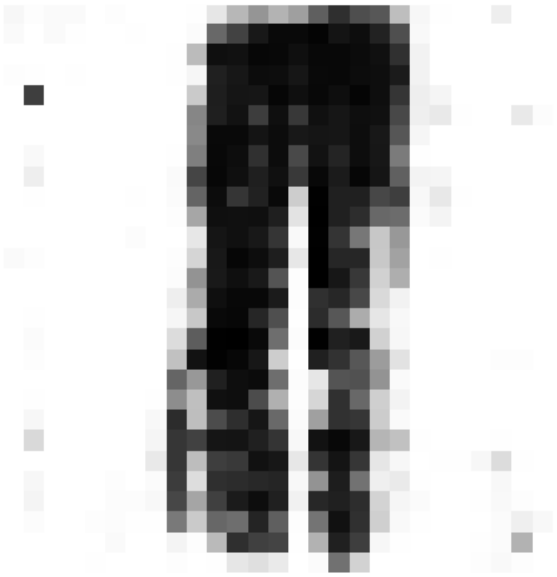

Figure 3: Image of Trouser generated from AC-GAN

h

Figure 4: Image of Dress generated from AC-GAN

h

Figure 5: Image of Sandal generated from AC-GAN

h

Figure 6: Image of Sneaker generated from AC-GAN

h

Figure 7: Image of Ankle Boot generated from AC-GAN

REFERENCES

[1] [n. d.]. deep learning - Pooling vs. stride for downsampling. ([n. d.]). https://stats.stackexchange.com/questions/387482/pooling-vs-stride-for-downsampling

[2] [n. d.]. Python Machine Learning - Third Edition. ([n. d.]). https://www.packtpub.com/product/python-machine-learning-third-edition/9781789955750

[3] [n. d.]. tf.keras.initializers.HeUniform | TensorFlow Core v2.4.1. ([n. d.]). https://www.tensorflow.org/api_docs/python/tf/keras/initializers/HeUniform

[4] 2016. An overview of gradient descent optimization algorithms. (Jan. 2016). https://ruder.io/optimizing-gradient-descent/

[5] Jason Brownlee. 2019. How to Develop a Conditional GAN (cGAN) From Scratch. (July 2019). https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/

[6] Jason Brownlee. 2019. How to Develop an Auxiliary Classifier GAN (AC-GAN) From Scratch with Keras. (July 2019). https://machinelearningmastery.com/how-to-develop-an-auxiliary-classifier-gan-ac-gan-from-scratch-with-keras/

[7] Jason Brownlee. 2019. How to use the UpSampling2D and Conv2DTranspose Layers in Keras. (June 2019). https://machinelearningmastery.com/upsampling-and-transpose-convolution-layers-for-generative-adversarial-networks/

[8] R. Gandhi, S. Nimbalkar, N. Yelamanchili, and S. Ponkshe. 2018. Plant disease detection using CNNs and GANs as an augmentative approach. In *2018 IEEE International Conference on Innovative Research and Development (ICIRD)*. 1–5. https://doi.org/10.1109/ICIRD.2018.8376321

[9] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]* (March 2015). http://arxiv.org/abs/1502.03167 arXiv: 1502.03167.

[10] Natsumi Kato, Hiroyuki Osone, Kotaro Oomori, Chun Wei Ooi, and Yoichi Ochiai. 2019. GANs-based Clothes Design: Pattern Maker Is All You Need to Design Clothing. In *Proceedings of the 10th Augmented Human International Conference 2019 (AH2019)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3311823.3311863

[11] Salome Kazeminia, Christoph Baur, Arjan Kuijper, Bram van Ginneken, Nassir Navab, Shadi Albarqouni, and Anirban Mukhopadhyay. 2020. GANs for medical image analysis. *Artificial Intelligence in Medicine* 109 (Sept. 2020), 101938. https://doi.org/10.1016/j.artmed.2020.101938

[12] Danqing Liu. 2017. A Practical Guide to ReLU. (Nov. 2017). https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7

[13] 2018 at 6:00pm Posted by L. V. on October 11 and View Blog. [n. d.]. Deep Learning Networks: Advantages of ReLU over Sigmoid Function. ([n. d.]). https://www.datasciencecentral.com/profiles/blogs/deep-learning-advantages-of-relu-over-sigmoid-function-in-deep

[14] Keras Team. [n. d.]. Keras documentation: Fashion MNIST dataset, an alternative to MNIST. ([n. d.]). https://keras.io/api/datasets/fashion_mnist/

[15] David Vint, Matthew Anderson, Yuhao Yang, Christos Ilioudis, Gaetano Di Caterina, and Carmine Clemente. 2021. Automatic Target Recognition for Low Resolution Foliage Penetrating SAR Images Using CNNs and GANs. *Remote Sensing* 13, 4 (Jan. 2021), 596. https://doi.org/10.3390/rs13040596 Number: 4 Publisher: Multidisciplinary Digital Publishing Institute.

[16] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]* (Sept. 2017). http://arxiv.org/abs/1708.07747 arXiv: 1708.07747.