**Generate Human Face using**

**Generative Adversarial Networks**

**Abstract**

The emergence of adversarial generative networks (GAN) has created completely new ways to transform and manipulate pixels into digital images. In this paper we will focus on comparing the results for various methods of generating faces using GAN. We will compare various GAN methods by changing parameters, we will use various discriminators and generators. We will use more databases and ore ways to process images. We will adapt various generators for the GAN training set. We will also explain some methods with examples. We will expose the methods with the good parts and the bad parts and we will check the accuracy for these methods. Based on extensive experimental study, we conduct the analysis on various data augmentation methods, observing how each affects the identification accuracy. Finally, we will pass the results in tables for better observation.

**1. Introduction**

Generative Adversarial Networks (or GAN’s) were first invented by Ian Goodfellow in 2014 and since then these models have seen an incredible burst of research improvements and applications. One of the coolest thinks about GAN’s is that you can take the underlying architecture but you can train the model on any dataset you want. Most famously researches at Nvidia have trained these models to generate faces leading to the popular website “ThisPersonDoesNotExist.com”. But we can take the same underlying model and you can train this on anything that we want. I mean we could train it on cats for example on or cars or what bedrooms, anything you have in terms of an image dataset you can use it to train this generative model. When you train a generative model on a dataset which is a fully unsupervised process, because you’re not using any labels, it turns out that these models actually discover the underlying structure in that dataset.

The idea behind GAN’s. In essence, what we have is two neural networks. We have the generator and we have the discriminator. The generator is in charge of doing the following: it gets a randomly sampled noise vector as an input. In most cases we’ll sample from a Gaussian distribution. We take that noise vector and we input it into the neural network and at the end we have an image. This image is then fed to a second neural network, the discriminator. The discriminator has one job: it needs to look at an image and decide whether that image comes, from the actual dataset, the real images that were training on or whether that image came from the generator and is this a fake image. Because we are controlling the training process, we have the label for each image that we feed to the discriminator, whether that image was a real one coming from the dataset or a fake image coming from the generator. By using this label, we can basically back propagate a training loss through this discriminator network in order to make it better. This generator network itself is also a fully differentiable neural network. If we stick these two networks back-to-back, we can actually back propagate the learning signal through this entire model pipeline. This way we can update with the same single loss function, we can update both the discriminator and the generator network until they both get really good at their job. The most important trick in this pipeline is to make sure that both these networks are well balanced during training, so that no one of them gets the upper hand. If we manage to do so and we train it for long enough then eventually what we’ll have is a generator that has been learning from the feedback of this discriminator network and can eventually we know manage to generate images that look very similar to the dataset. [2]

**2. State-of-the-art GAN techniques.**

The progressive growing of the layers in our generative model. This was an idea published by Nvidia in their paper called progressive growing [2] of GAN’s. The idea is: you basically start with a generative model that generate very small images of super low resolution and, at the same time, the discriminator also gets to discriminate between very low resolution images. This makes the entire process actually super simple so this network is very stable and it converges quickly. Once that network has stabilized, we can simply add an additional layer to both the generator and the discriminator architecture which works at a slightly higher resolution and you can keep on training. There are a few additional tricks where instead of just adding this layer one shot, you basically do it gradually by blending the previous layer towards the higher resolution one. But in practice this is what happens: your generator starts by generating very low resolution images and the discriminator discriminates then, and during the training process you gradually scale up the resolution of the images in your training pipeline. The second very influential paper by Nvidia in the generative model landscape was a model architecture called style GAN. [2]In fact, style GAN is the model that we’re going to be using to manipulate our images. Traditionally what happens is that you take a generator Architecture and it gets a random noise sample as an input. You feed that noise sample through a whole bunch of up sampling and convolutional layers until you get an image. What the style GAN generator does is slightly different. First it has a mapping network. This mapping network takes the noise vector, Z, and transforms it into a different vector called W. The important thing here is that the W vector doesn’t have to be Gaussian anymore. The distribution of those W can be whatever the generator wants it to be. Then the actual generator architecture doesn’t start from a random noise vector anymore. It starts from a constant vector and this constant vector is actually optimized during training. It’s kind of like a seed, a fixed seed in the beginning of the first layer of the generator architecture. But the actual numbers of that seed, the values of that vector will they’re constant but they are optimized during the training process. Finally, the output of the mapping layer, W is plugged into multiple layers of the generative architecture using a blending layer. In the during also we add noise to these parameters. Imagine that you have a dataset right and imagine we look at two properties.[2] We look at gender and we look at facial hair. We have male and female and we beard and no beard. In most image datasets of people, you will find very little women that have beards. So, in other words, our data distribution has a gap there. If we’re sampling from a Gaussian distribution, that distribution is uniform and doesn’t have any gaps. So, this is essentially what this mapping network allows you to do. It allows you to sample from a uniform distribution but then warp that distribution in such a way that you can have gaps, for example, where there are no actual images. The idea is that if this warping of the space is already done by the mapping network, so your W vector is already in a good shape, then your actual generator who takes that vector and turns it into an image has a much simpler job of doing that, because the relationship between images and the input vector is more one-to-one. It doesn’t have any gaps or strange distortions that it has to learn. In the style GAN paper [], they apply this generative architecture, but they also use the progressively growing of the layers. With those two tricks combined, they were able to create a very powerful GAN, that was able to produce incredibly realistic images. [2]

**3.** **Style GAN’s latent space**

The general principle that we're gonna leverage here is that when you train a generative model, the latent space usually learns the underlying structure of your dataset. That structure is learned fully unsupervised by the generative model because we’re not using any labels in the dataset. Instead of manipulating images in the pixel domain, let’s manipulate them in the latent space of that generative model. In order to do this, starting from any given image, we’re gonna have to find a way to find a query image inside the latent space of the generator. More specifically, we start with a image of, for example, Barack Obama. The question is how can we find the latent vector Z such that if we send Z through the generator, we get this image of Barack Obama. We can try randomly sampling a whole bunch of these latent vectors and see which is closest but that’s gonna take a really long time. One of the most straightforward things that you could try is given the fact that this generative model is a fully differentiable neural network. We can basically send gradients through it, we could basically randomly start from any latent vector Z, we can generate a random image but then we compare that image to the query image of Barack Obama. We could define a simple loss function, the pixel by pixel difference, the L2 difference between these two images and we can do gradient descent, not on the image but we send our gradients through this generator model and we actually update the latent vector Z at the beginning of our generator. By applying gradient descent on this L2 pixel loss we could, in theory, find the optimal latent vector Z that gives us a good image of Barack Obama. [2]Unfortunately this doesn’t really work. The L2 optimization objective is going to start going in the direction of Barack Obama but before it gets there it’s gonna get stuck in a very bad local minimum of an image that doesn’t look at all like Barack Obama and once we’re , the optimization objective simply can’t get out anymore. So L2 optimization directly in the pixel space doesn’t work. We will find a different approach. We can use a pre trained image classifier as a lens to look at our pixels. So rather than optimizing our L2 loss directly in the pixel space, we’re gonna send both the output of our generator and the query image through a pre-trained VGG network that was trained to classify image net images. But instead of actually going all the way to the last layer into the classification, we’re gonna cut off the head of that network and we’re gonna simply extract a feature vector somewhere inside the last fully connected layers of that classifier. So we send these images through the pre trained classifier. We extract a feature vector at some of the last fully connected layers in the network and this gives us a high-level semantic representation of what is in the image. It turns out that if we actually do gradient descent on this feature vector rather than on the pixels of the image or approach does work. There is one final problem with this approach and that is that this is actually really slow. It takes a very long time for the optimizer to actually find a good latent code. We need to make a dataset. First we’ll sample a whole brunch of random vectors, we’ll send them through the generator and will generate faces. Once we have that dataset we can train a ResNet to go from the images to their respective label code. In the repo we’ll be using in a notebook there is already a pre trained ResNet like this. So we can just use it out of the box. So here’s what our pipeline currently looks like. We take a query image. We send it through the residual Network and this network gives us an initial estimate of the latent space vector in the Style GAN Network. We then take that latent vector. We send it through the generator which gives us an image. On this image We apply a pre trained VGG Network in order to extract features from it. We do the same thing for our query image of Barack Obama. Then in that feature space we start doing gradient descent.[2] We minimize that L2 distance in this feature space and we send those gradients through the generator model all the way back into our latent code. Importantly, during this optimization process the generator weights itself are completely fixed. The only thing we’re updating is the latent code at the input of our generator. In this optimization process there’s a lot of different things that we can tweak. For example, which specific layer of the VGG network are you using as that semantic feature vector. Or, for example, are we applying a mask to the face, such that only pixels within the face region are actually used to compute that L2 difference. We can even add a penalty on the latent code that we’re optimizing, such that it doesn’t move to for away from the concept of a face according to the Style GAN Network. Because as it turns out, we can use this process to find any image we want inside the Style GAN Network. Even a Style GAN trained on faces, by doing this approach, we could basically find a car inside the latent space of Style GAN. But the problem is that the vector which gives us a car is going to be very far away from that Gaussian distribution that we actually started from. If we want to start manipulating this face it’s gonna be a really good idea to make sure that whatever latent vector we’re gonna find is going to be similar to the concept of a face inside this Style GAN Network. [2]

Here's the entire process. We start with a query image. We send it through the ResNet and we will get a initial estimate and then we do this latent space optimization until we finally get our optimized image which is a close as possible to the query image that we started from.

**4. GAN using Co-occurrence Matrices**

Since the seminal work on GAN’s, there are many papers on exploitation GAN’s to get pictures. These works focus in generating pictures of high sensory activity quality, image-to-image translations, domain transfer, super-resolution, image synthesis and completion, and generation of facial attributes and expressions. many ways are projected in the area of image forensics over the past years. Recent approaches have targeted on applying deep learning primarily based methods to find tampered pictures. The detection of GAN pictures may be a new space in image forensics and there area unit only a few papers during this space. Connected fields additionally embrace detection of laptop generated (CG) pictures. The foremost relevant work may be a recent paper on police work GAN primarily based image-to image translation generated exploitation Cycle GAN. Here the authors compare numerous existing ways to spot Cycle GAN images from traditional ones. The highest results they obtained exploitation a combination of residual options and deep learning. Similar to, the authors in cypher the residuals of high pass filtered pictures and so extract co-occurrence matrices on these residuals, that area unit then concatenated to make a feature vector which will distinguish real from pretend GAN pictures. In distinction to these approaches, our approach doesn't would like any image residuals to be computed. Rather, our technique directly computes co-occurrence matrices on the 3 color channels that area unit then passed through a deep convolutional neural network (CNN) to learn a model which will find pretend GAN generated pictures. [1]

To sight GAN pictures, we tend to reason co-occurrence matrices on the RGB channels of a picture. Co-occurrence matrices have been antecedently employed in steganalysis to spot pictures that have knowledge hidden in them and in image forensics to detect or localize tampered pictures. Co-occurrence matrices area unit sometimes computed on image residuals by passing the image through several filters and getting the distinction. Typically options or statistics also are computed on the matrices and so a classifier is trained on these options to classify knowledge hidden or tampered pictures. [1]  
However, we tend to reason co-occurrence matrices directly on the image pixels on every of the red, inexperienced and blue channels and pass them through a convolutional neural network, thereby permitting the network to find out vital options from the co-occurrence matrices. Specifically, the primary step is to reason the co-occurrence matrices on the RGB channels to get a 3x256x256 tensor. This tensor is then felt a multi-layer deep convolutional neural network: conv layer with thirty two 3x3 convs + ReLu layer + conv layer with thirty two 5x5 convs + goop pooling layer + conv layer with sixty four 3x3 convs + ReLu layer + conv layer with sixty four 5x5 convs + max pooling layer + conv layer with 128 3x3 convs + ReLu layer + conv layer with 128 5x5 convs + goop pooling layer + 256 dense layer + 256 dense layer + sigmoid layer. [1]

**4.1. Experiments**

We evaluate this method on two diverse and challenging datasets which contain GAN generated images such facial attributes and expressions.

**4.2. Datasets**

Cycle GAN dataset contains unmatched image to image translation of varied objects and scenes like horse to zebras, summer to winter, pictures to paintings, for example Monet, Van Gough, vogue transfer like labels to facades and others that were generated using a cycle consistent GAN framework [3]. We tend to followed the directions provided by the authors as elaborated in https://github.com/junyanz/pytorch-CycleGAN-andpix2pix , and obtained 36302 pictures (18151 GAN and 18151 non GAN pictures). The distribution of GAN pictures are as follows: there are 2014 pictures of apple to orange, 2401 pictures of horse to zebra, 2193 pictures summer to winter, 2975 pictures cityscapes, 400 pictures facades, 1096 pictures map to sat, 1500 pictures Ukiyoe, 1500 pictures of Vincent van Gogh, 1500 pictures of Paul Cezanne, 2752 pictures of Monet. We tend to obtain this distribution from the authors and we compare our approach with their results in the Experiments section.

Star GAN dataset contains of 19990 pictures of which 1999 were from faces taken from the celebA dataset and the remaining 17991 images were GAN generated images with 5 varying facial attributes like brown hair, black hair, blond hair, aged, gender modification, aged and 4 combinations of the same. We followed the direction in https://github.com/yunjey/stargan , to generate the images.

**4.3. Evaluation**

We first appraise our approach on 2 datasets on an individual basis and then perform cross evaluation on the 2 datasets (one dataset as training and alternative as testing) to visualize the generalizability of our approach. For each dataset: 50% of the info is employed for training, 25% for validation and 25% for testing. We have a tendency to train the network for 50 epochs with a batch size of 40 and use a variant to adjustive random gradient as optimizer. We have a tendency to obtained a high coaching and validation accuracy of 99.90% and 99.40% on Cycle GAN dataset and 99.43% and 99.39% on Star GAN dataset. We then evaluated the model on the held-out test sets and obtained a testing accuracy of 99.37% on the Star GAN dataset and 99.71% on the Cycle GAN dataset.

|  |  |  |
| --- | --- | --- |
| Training dataset | Testing dataset | Accuracy |
| Star GAN | Cycle GAN | 93.42 |
| Cycle GAN | Star GAN | 99.49 |

Next, we evaluate the generalizability of our approach by testing on one dataset and training on the opposite. Initial we tend to train on all the picture with the Cycle GAN dataset (35302 images) and test the model on all pictures of the Star GAN dataset (19990 images), and then we reverse the experiment where we train on Star GAN and test on Cycle GAN. We train the network until 50 epochs and report on the model that gave the best accuracy. Our technique still maintains a high accuracy even across diverse datasets. The model trained on Cycle Gan dataset incorporates a higher accuracy of 99.45% as compared with the model trained on Star GAN dataset that got 93.42%. The lower accuracy for the model trained on Star GAN dataset can be as a result of non-uniform distribution of class samples in the Star GAN dataset, and because of the various image sources/categories within the Cycle GAN dataset. [3]

**4.4 Comparison with State-of-the-art**

We compare our approach with the results given in [1]. Here the authors conduct a study on detection of pictures manipulated by GAN based image to image translation on the Cycle GAN dataset. For analysis, they adopt a leave one manipulation out strategy on the classes within the Cycle Gan dataset, wherever at every iteration images belonging to one class are unit put aside for validation and the pictures from other categories are used for training. The methods evaluated are based on steganalysis, generic image manipulations, detection of computer graphics, a GAN discriminator used in the Cycle GAN paper, and generic deep learning design pretrained on ImageNet, however fine tuned to the Cycle GAN dataset. Among these the highest preforming ones were from high-pass residual pictures, a deep neural network designed to extract residual features and XceptionNet deep neural network trained on ImageNet but fine-tuned to the present dataset. We report the result as mentioned in the paper only from these three methods and compare with our approach.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Method | Ap2or | Ho2zeb | Wint2sum | Citysc. | facades | Map2sat | Ukiyoe | Van Gogh | Cezanne | Monet | Average |
| Steganalysis feat. | 98.93 | 98.44 | 66.23 | 100.00 | 97.38 | 88.09 | 97.93 | 99.73 | 99.83 | 98.52 | 94.40 |
| Cozzalino2017 | 99.90 | 99.98 | 61.22 | 99.92 | 97.25 | 99.59 | 100.00 | 99.93 | 100.00 | 99.16 | 95.07 |
| XceptionNet | 95.91 | 99.16 | 76.74 | 100.00 | 98.56 | 76.79 | 100.00 | 99.93 | 100.00 | 95.10 | 94.49 |
| Proposed | 99.78 | 99.75 | 99.72 | 92.00 | 80.63 | 97.51 | 99.63 | 100 | 99.63 | 99.16 | 97.84 |

Results on original images: Here we consider the photos generated from the Cycle GAN dataset as it is and don’t perform any post processing on the photos. In the table above, we summarizes the results of our projected approach along with the three top performing approaches in [1]. On average, our technique out performed other methods and was able to achieve an accuracy of 97.84. On most categories, our approach was higher than or on-par with the opposite top methods. These can be as a result of the original pictures n these classes were JPEG compressed which could have affected the classification accuracy. Next, we study the effect of compression on the method.

|  |  |  |
| --- | --- | --- |
| JPEG Quality Factor | Trained on original images | Trained on JPEG compressed images |
| 95 | 74.5 | 93.78 |
| 85 | 69.46 | 91.61 |
| 75 | 64.46 | 87.31 |

Effect of JPEG compression: In [1], the authors investigated the sensitivity of their strategies on compression. Employing a compression methodology the same as Twitter, they trained their detection methods on original uncompressed pictures and tested on JPEG compressed pictures. The target of this study was to check the robustness of the detection techniques when pictures are posted in social networks such as Twitter. In the second scenario, they train and test on the JPEG compressed images. We performed both of these experiments on the Cycle GAN dataset. Since we are not aware of the exact JPEG quantization tables used in Twitter, our approach was similar but we tested on three different JPEG quality factors (QF): 95, 85 and 75. We used 50% of the data for training, 25% for validation and 25% for testing. The results are reported on the 25% testing data of the Cycle GAN dataset (9,076 images). As shown in Tab. 3, the accuracy progressively drops as the QF decreases from 95-75, when trained on the original images. But when the JPEG compressed images are used for training, the accuracy shows a substantial increase. Even at a QF of 75, the accuracy is still 87.31%. This is also consistent with the results reported in [1], where they report close to a 10% drop in accuracy on Twitter-like compressed pictures.

**5. Face Generation for Low-shot Learning**

We use the dataset of MS-Celeb-1M Challenge-2: Lowshot learning for training face identification network. This dataset is divided into two sets (base set and novel set. It is important to note that there is no overlap between base set and novel set. [4] Our goal is to identify individuals in novel set, while keeping the recognition accuracy using base set. In attribute boosting using GAN, additional dataset, CelebA, is utilized to extract attribute variables. CelebA contains more than 200000 images with 40 attribute annotations for about 10000 identities. Entire database is used to extract attributes values. Note that, this dataset is used only for attribute boosting.

|  |  |  |  |
| --- | --- | --- | --- |
| MSCeleb-Database | Subjects | Training set | Test set |
| Base set | 20000 | 50-100 | 5 |
| Novel set | 1000 | 5 | 20 |

|  |  |  |  |
| --- | --- | --- | --- |
| CelebA | Subjects | Attributes | Total |
| CelebA | 10177 | 40 | 202599 |

[5]

Reconstruction. We reconstruct face images using VAE/GAN and BEGAN, respectively. We can observe the discrepancy between the reconstructed faces from VAE/GAN and original input faces. Especially, some results does not look like human faces. Meanwhile, BEGAN produces better results, more human-like faces, and also the reconstructed faces more similar to original input faces. As a result, we decided to choose BEGAN for the face generation model. [5] To enhance the detail in reconstructed faces, we add skip connections in decoder and generator of BEGAN. Note that all the generated faces for identification in this paper are synthesized using BEGAN with skip connection. Before the training section, each data is pre processed to fit into our face identification network. [4]We have a tendency to cropped into a square with the sides size of shorter side of the first image. Pictures were cropped on the middle. Then, we have a tendency to roughly align pictures supported eye location and resize pictures into 224x224x3.

Our final face identification network establish people in novel set, whereas keeping recognizable subject in base set. Like verification network, face identification network utilizes design and coaching strategy. it's necessary to notice that, the sole distinction between 2 networks is fine-tuning information. Face identification network is pre-trained with based mostly set and so fine-tuned with base set, novel set and our extra increased novel set. For analysis, we have a tendency to measure accuracy, precision and coverage. Suppose that N pictures are available in the test set, M pictures are evaluated, and C pictures are recognized properly. Then, the accuracy, precision and coverage are defined as C/N, C/M and M/N. The coverage is reported when the precision is at 99%. [4]

We gradually increase novel set with planned augmentation methods and investigate the changes of identification accuracy. As a result of the quantity of base set is considerably larger than that of novel set, baseline network trained with original base set and novel set and rarely identifies novel set. It is a result of the importance of individual is proportional to the number of training pictures. Therefore, the accuracy and coverage of novel set are increased by adding increased data. From these experiments, we observe that both augmentation methods, classical augmentation and face generation using GAN, improve identification performance. Considering its importance in performance, pose transition and attribute boosting using BEGAN are more effective in improving performance than the classical method. Notice that accuracy improves from 17.40% to 40.40% for 1-shot scenario and from 57.42% to 80.45% for 3-shot scenario. Likewise, coverage at 99% precision is advanced from 0.02% to 1.75% for 1-shot scenario and from 0.22% to 52.07% in 3- shot scenario. The results demonstrate that face generation with GAN certainly complement lack of intra-class variation. [4]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Training method | Test set | 1-shot | | 3-shot | |
| Accuracy | Coverage | Accuracy | Coverage |
| BS+NS | Base set | 98.70 | 99.00 | 98.68 | 98.95 |
| Novel set | 0 | 0.02 | 0 | 0.02 |
| BS+NS+CS | Base set | 98.43 | 98.97 | 98.08 | 98.42 |
| Novel set | 17.40 | 0.02 | 57.42 | 0.22 |
| BS+NS+CS+GS | Base set | 97.00 | 94.47 | 95.90 | 89.50 |
| Novel set | 40.00 | 1.75 | 80.45 | 52.07 |

The table below shows them the result of manipulating the ratio real – fake of augmented data. Novel set, attribute and verified datasets using verification network are denoted by NS, attr and V each. [4]

|  |  |  |
| --- | --- | --- |
| Training Method | 1-shot | |
| Accuracy | Coverage |
| NS + pose(1 – 11)  +V pose( 1 – 3)  + 1 – 1  + 10 – 1  + 20 – 1 | 29.37  30.20  39.80  49.53  50.72 | 0.0095  1.95  3.20  10.65  10.12 |
| NS + attr(1 - 40)  + V attr(1 – 13)  + 1 – 1  + 10 – 1  + 20 – 1 | 24.55  25.45  39.35  49.57  49.60 | 0.40  0.85  6.42  7.60  11.17 |

[4]

**6. Conclusion**

Image generation techniques are attractive in various computer vision applications, especially because of the difficulties of labeled data collection for training. Among those, we attempted to generate face images with several attributes and poses using GAN, enlarging the novel set to achieve increased performance on low-shot face recognition task. We also proposed a novel method to detect GAN generated fake images using a combination of pixel co-occurrence matrices and deep learning. Co-occurrence matrices are computed on the color channels of an image and then trained using a deep convolutional neural network to distinguish GAN generated fake images from real ones. We have experimented with these methods on several datasets.

**7. Bibliography**

[1] F. Marra, D. Gragnaniello, D. Cozzolino, et al., “Detection of gangenerated fake images over social networks,” in 2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR), 384–389, IEEE (2018)

[2] [Tero Karras](https://arxiv.org/search/cs?searchtype=author&query=Karras%2C+T), [Samuli Laine](https://arxiv.org/search/cs?searchtype=author&query=Laine%2C+S), [Timo Aila](https://arxiv.org/search/cs?searchtype=author&query=Aila%2C+T). “A Style-Based Generator Architecture for Generative Adversarial Networks”, 29 Mar 2019, Cornell University

[3] J.-Y. Zhu, T. Park, P. Isola, et al., “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in IEEE International Conference on Computer Vision, (2017).

[4] Choe, J., Park, S., Kim, K., Hyun Park, J., Kim, D., & Shim, H. (2017). Face generation for low-shot learning using generative adversarial networks. In Proceedings of the IEEE International Conference on Computer Vision Workshops (pp. 1940-1948).

[5] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In 2015 IEEE International Conference on Computer Vision (ICCV), pages 3730–3738, Dec 2015.