

## PROJECT

# Generate Faces

Intactic Work here!! This is a great first submission. You have a vivid grasp of DCGAN concepts, I've suggested a few tips which will help you generate very realistic faces and you'll pass this opics as well as this hanodegree.  oking forward to you'r next submission  equired Files and Tests  The project submission contains the project notebook, called 'dind, face, generation.jpynb''.  All the unit tests in project have passed.  Unid the Neural Network  The function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the discriminator is as a sequence of core layers with the following points to note:  Avesome  • implemented isternanator using cancal * strides to avoid making sparse gradients instead of max-pooling layers.  • used Leaby, relai instead of ReLU for the same reason of avoiding sparse gradients as leaby, relai progradients on the value of the same reason of avoiding sparse gradients as leaby, relai instead of as output layer implemented Batch Norm to avoid "internal covariate shift".  Tips  • Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent focal minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low leng-rate in discriminator as mentioned here.	A part of the Deep Learning Nanodegree Foundation Program
NOTES  NASE YOUR ACCOMPLISHMENT:   Prequires Changes  PRECIFICATIONS REQUIRE CHANGES  Internation with the heart This is a great first pubmission. You have a vivid grasp of DCGAN concepts. The suggested a few tips which will help you generate very realistic faces and you'll pass this great and well as this horseldigne.  Selding forward to your next submission @  Required Files and Tests  The project submission contains the project notebook, called "dind, face, generation jaynth".  All the unit tests in project have passed.  Build the Neural Network  The function model inputs is implemented correctly.  The function discriminator of find I for the same reason of modeling sparse gradients instead of max pooling layers.  I may be included a find of field I for the same reason of modeling sparse gradients and fluxly rule allows gradients on flow bedwards unimpreded.  I may be included in institution for a good find a find in a same reason of modeling sparse gradients and fluxly rule allows gradients on flow bedwards unimpreded.  I may be included in institution for a good find grade in the pure institution of the gradients are instituted for the same reason of modeling sparse gradients and fluxly rule allows gradients on flow bedwards unimpreded.  I may be instituted in the same reason of modeling sparse gradients and fluxly rule allows gradients on flow bedwards unimpreded.  I may be instituted in the same reason of modeling sparse gradients and fluxly rule allows gradients on flow bedwards unimpreded.  I may be instituted to the same reason of modeling sparse gradients and states the symmetry and thus, help converge faster as well as prevent local minima. Also, this institute in finding sparse gradients are gradients are in all layers.  I implement dropouts with local draw far gradients mentioned bees.	PROJECT REVIEW
HARE YOUR ACCOMPLISHMENT IN INC.  Equires Changes  ***SECTION AND REQUIRE CHANGES  ***THERETHS IS a great first submission. You have a vivid grasp of DCGAN concepts. I've suggested a few sps which will help you generate very realists: faces and you'll past this supplements which will help you generate very realists: faces and you'll past this supplements to your meet submission @  ***Equired Files and Tests  ***The project submission contains the project notebook, called 'dind, face, generation.lpynh'.  ***All the unit tests in project have passed.  ***All the unit tests in project have passed.  ***All the unit tests in inplemented correctly.  ***The function model_inputs is implemented correctly.  ***The function model_inputs is implemented correctly.  ***Note: job of implementing the discretanater is as a sequence of conv. [syers with the following points to note:  ***Awesome**  *** Implemented acceptances unline [conc2] *   straces to avoid making sparse gradients instead of man-sooling layers.  *** or of [-insyy-a] instead of like I/b or the same reason of avoiding sparse gradients as [-insyy-a] allows gradients to flow backwards unimpeded.  *** Implemented distribution to accord long like to below understand what it's) to break the symmetry and thus, help converge faster as well as provent local minima. Also, this institutes is designed to keep the scale of the gradients roughly the same in all layers.  *** Insplemented distribution of good doing like to below understand what it's) to break the symmetry and thus, help converge faster as well as provent local minima. Also, this institutes is designed to keep the scale of the gradients roughly the same in all layers.  *** Insplemented distribution of good doing the scale in a first food of the scale acceptance.  *** Insplemented distribution of good doing the scale in a first food of the scale acceptance.  *** Insplemented distribution of good doing the scale in a first food of the scale acceptance.  *** Insplemented distribution of good doing the scale	CODE REVIEW
Executions in course changes  SPECHEATORS REQUISE CHANGES  Handsit: Work herell This is a great first submission. You have a whild greap of DCGAN concepts. I've suggested a few tips which will help you generate very realistic faces and you'll pass this operation as well as the Nanodegree.  Color forward to your next submission @  equired Files and Tests  The project submission contains the project notebook, called "dind, face generation.ipynth".  All the unit tests in project have passed.  Little the Neural Network  The function model Linputs is implemented correctly.  Whice job of implementing the discrisionator is implemented correctly.  Whice job of implementing the discrisionator is implemented correctly.  Avesome  In implementing discrisionator is discrisionator in grower in granted of evoluting sparse gradients as jussing, retain allows gradients to flow backwards unimpeded.  Little Section of the submission of the project notebook, called "dind, face generation.ipynth".  **This is a submission contains the project notebook, called "dind, face generation.ipynth".  **Little Handson discriminator is implemented correctly.  **We job of implementing the discrisionator is implemented correctly.  **We job of implementing the discrisionator in a sequence of cow layers with the following points to note:  **Avesome  In implemented gisscrimator in the discrisionator in a sequence of interest to avoid making sparse gradients as jussing, retain allows gradients to flow backwards unimpeded.  Little Section discriminator in the properties of the gradients required to a sequence of general covariate shift.  **This is the function generator is implemented correctly.  **The function model Lios is implemented corr	NOTES
Executions in course changes  SPECHEATORS REQUISE CHANGES  Handsit: Work herell This is a great first submission. You have a whild greap of DCGAN concepts. I've suggested a few tips which will help you generate very realistic faces and you'll pass this operation as well as the Nanodegree.  Color forward to your next submission @  equired Files and Tests  The project submission contains the project notebook, called "dind, face generation.ipynth".  All the unit tests in project have passed.  Little the Neural Network  The function model Linputs is implemented correctly.  Whice job of implementing the discrisionator is implemented correctly.  Whice job of implementing the discrisionator is implemented correctly.  Avesome  In implementing discrisionator is discrisionator in grower in granted of evoluting sparse gradients as jussing, retain allows gradients to flow backwards unimpeded.  Little Section of the submission of the project notebook, called "dind, face generation.ipynth".  **This is a submission contains the project notebook, called "dind, face generation.ipynth".  **Little Handson discriminator is implemented correctly.  **We job of implementing the discrisionator is implemented correctly.  **We job of implementing the discrisionator in a sequence of cow layers with the following points to note:  **Avesome  In implemented gisscrimator in the discrisionator in a sequence of interest to avoid making sparse gradients as jussing, retain allows gradients to flow backwards unimpeded.  Little Section discriminator in the properties of the gradients required to a sequence of general covariate shift.  **This is the function generator is implemented correctly.  **The function model Lios is implemented corr	
And the function model imputs is implemented correctly.  The function model institution to a sequence of conv. layers with the following points to note:  Assessme  In implemented discritations using consider the same reason of avoiding sparse gradients instead of max-publing layers.  In used layers also contains the project note and institution to avoid minimal accordance white.  It is used layers also contains the project note and consider the same reason of avoiding sparse gradients as layers and thus, help converge faster as well as prevent local minima. Also, this initialization is designed to keep the scale of the gradients roughly follow the same tips of using dropouts and weight initialization as that for the latescriature.  The function generator is implemented correctly.  The function model is the same reason of avoiding sparse gradients in lates, the property and thus, help converge faster as well as prevent local minima. Also, this initialization is designed to keep the scale of the gradients roughly the same in all layers.  It is prevented to serve the scale of the gradients roughly the same in all layers.  It implement dropouts with low gray_rate in east-created as memory and thus, help converge faster as well as prevent local minima. Also, this initialization is designed to keep the scale of the gradients roughly the same in all layers.  It implement dropouts with low gray_rate in east-created is a mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialization as that for the latescriaturator.	
plet at well as the Nanodegree.  deling forward to your nest submission   equired Files and Tests  The project submission contains the project notebook, called "dind_face_generation.ipynb".  All the unit tests in project have passed.  Unit the Neural Network  The function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the _discriminator is implemented correctly.  Nice job of implementing the _discriminator as a sequence of _cow_layers with the following points to note:  Avecome  implemented discriminator using _compat + trates_ to avoid making sparse gradients instead of max-pooling layers.  used _laway_retup   instead of fletul for the same reason of avoiding sparse gradients as _lexy_retup   allows gradients to flow backwards unimpeded.  used _layers as output layer  implemented Backhrom to avoid "internal covariate shift".  Tips  Use Xavler weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this mitisligner is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low   dreg_rete   in _discriminator   as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialization as that for the _discriminator.  The function model_loss is implemented correctly.	SPECIFICATIONS REQUIRE CHANGES
The project submission contains the project notebook, called "dind_face generation.ipynb".  All the unit tests in project have passed.  Little unit tests in project have passed.  Little unit tests in project have passed.  Little function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Little function discriminator is implemented of REU for the same reason of avoid making sparse gradients instead of max-pooling layers.  Little function discriminator is implemented of REU for the same reason of avoiding sparse gradients as lessly_relay allows gradients to flow backwards unimpeded.  Little function avoid "instead and the production of the gradients roughly the same in all layers.  Little function generator is implemented correctly.  Little function generator is implemented correctly.  Little function generator is implemented correctly.  Little function model_loss is implemented correctly.	intastic Work here!! This is a great first submission. You have a vivid grasp of DCGAN concepts. I've suggested a few tips which will help you generate very realistic faces and you'll pass this oject as well as this Nanodegree.
The project submission contains the project notebook, called "dind_face_generation.ipynb".  All the unit tests in project have passed.  Unit the Neural Network  The function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the   discriminator is a sequence of   cow   layers with the following points to note:  Avesome  Implemented   discriminator is implemented   storage   s	oking forward to your next submission 😄
All the unit tests in project have passed.  uild the Neural Network  The function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the _discristinator   as a sequence of _cow_layers with the following points to note:  Avesome  • implemented _discristinator   using _comed   *   strides   to avoid making sparse gradients instead of max-pooling layers. • used _lasky_reu   instead of RetU for the same reason of avoiding sparse gradients as _lasky_reu   allows gradients to flow backwards unimpeded. • used _signed a so output layer • implemented BatchNorm to avoid "internal covariate shift".  Tips  • Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers. • Implement dropouts with low _drop_rate   in _discrissinator   as mentioned here.  The function generator is implemented correctly.  The function model_loss is implemented correctly.	equired Files and Tests
All the unit tests in project have passed.  uild the Neural Network  The function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the _discristinator   as a sequence of _cow_layers with the following points to note:  Avesome  • implemented _discristinator   using _comed   *   strides   to avoid making sparse gradients instead of max-pooling layers. • used _lasky_reu   instead of RetU for the same reason of avoiding sparse gradients as _lasky_reu   allows gradients to flow backwards unimpeded. • used _signed a so output layer • implemented BatchNorm to avoid "internal covariate shift".  Tips  • Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers. • Implement dropouts with low _drop_rate   in _discrissinator   as mentioned here.  The function generator is implemented correctly.  The function model_loss is implemented correctly.	
The function model inputs is implemented correctly.  The function discriminator is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the discriminator as a sequence of cow layers with the following points to note:  Awesome  implemented discriminator using cow2d   strides to avoid making sparse gradients instead of max-pooling layers.  used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.  used signed as output layer  implemented BatchNorm to avoid "internal covariate shift".  Tips  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model loss is implemented correctly.	The project submission contains the project notebook, called "dlnd_face_generation.ipynb".
The function model inputs is implemented correctly.  The function discriminator is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the discriminator as a sequence of cow layers with the following points to note:  Awesome  implemented discriminator using cow2d   strides to avoid making sparse gradients instead of max-pooling layers.  used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.  used signed as output layer  implemented BatchNorm to avoid "internal covariate shift".  Tips  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model loss is implemented correctly.	
The function model_inputs is implemented correctly.  The function discriminator is implemented correctly.  Nice job of implementing the _discriminator_ as a sequence of _conv_layers with the following points to note:  AWESOME  • implemented _discriminator_ using _conv2d + _strides_ to avoid making sparse gradients instead of max-pooling layers. • used _lasaby_realu_ instead of ReLU for the same reason of avoiding sparse gradients as _lasaby_realu_ illows gradients to flow backwards unimpeded. • used _staged_is as output layer • implemented BatchNorm to avoid "internal covariate shift".  Tips  • Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers. • Implement dropouts with low _drop_rate in _discriminator_is mentioned here.  The function generator is implemented correctly.  The function model_loss is implemented correctly.	
The function discriminator is implemented correctly.  Nice job of implementing the discriminator as a sequence of converge layers with the following points to note:  AWESOME  Implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers.  used lasty_retu instead of RetU for the same reason of avoiding sparse gradients as leaky_retu allows gradients to flow backwards unimpeded.  used lasty_retu instead of RetU for the same reason of avoiding sparse gradients as leaky_retu allows gradients to flow backwards unimpeded.  used lasty_retu instead of RetU for the same reason of avoiding sparse gradients as leaky_retu allows gradients to flow backwards unimpeded.  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.	uild the Neural Network
The function discriminator is implemented correctly.  Nice job of implementing the discriminator as a sequence of converge layers with the following points to note:  AWESOME  Implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers.  used lasty_retu instead of RetU for the same reason of avoiding sparse gradients as leaky_retu allows gradients to flow backwards unimpeded.  used lasty_retu instead of RetU for the same reason of avoiding sparse gradients as leaky_retu allows gradients to flow backwards unimpeded.  used lasty_retu instead of RetU for the same reason of avoiding sparse gradients as leaky_retu allows gradients to flow backwards unimpeded.  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.	
Avesome  Implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers.  Used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.  Used signoid as output layer  Implemented BatchNorm to avoid "internal covariate shift".  Tips  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.	The function model_inputs is implemented correctly.
Avesome  Implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers.  Used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.  Used signoid as output layer  Implemented BatchNorm to avoid "internal covariate shift".  Tips  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.	
• implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers. • used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded. • used signoid as output layer • implemented BatchNorm to avoid "internal covariate shift".  Tips  • Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers. • Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	The function discriminator is implemented correctly.
implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers.  used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.  used sigmoid as output layer  implemented BatchNorm to avoid "internal covariate shift".  Tips  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	Nice job of implementing the discriminator as a sequence of conv layers with the following points to note:
implemented discriminator using conv2d + strides to avoid making sparse gradients instead of max-pooling layers.  used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.  used sigmoid as output layer  implemented BatchNorm to avoid "internal covariate shift".  Tips  Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	Awesome
<ul> <li>used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.</li> <li>used sigmoid as output layer</li> <li>implemented BatchNorm to avoid "internal covariate shift".</li> </ul> Tips <ul> <li>Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.</li> <li>Implement dropouts with low drop_rate in discriminator as mentioned here.</li> </ul> The function generator is implemented correctly. The function model_loss is implemented correctly. The function model_loss is implemented correctly.	
<ul> <li>implemented BatchNorm to avoid "internal covariate shift".</li> <li>Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.</li> <li>Implement dropouts with low drop_rate in discriminator as mentioned here.</li> </ul> The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.  The function model_loss is implemented correctly.	• used leaky_relu instead of ReLU for the same reason of avoiding sparse gradients as leaky_relu allows gradients to flow backwards unimpeded.
Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	
Use Xavier weight initialization (A good blog link to help understand what it's) to break the symmetry and thus, help converge faster as well as prevent local minima. Also, this initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	
initializer is designed to keep the scale of the gradients roughly the same in all layers.  Implement dropouts with low drop_rate in discriminator as mentioned here.  The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	
The function generator is implemented correctly.  Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	
Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	Implement dropouts with low drop_rate in discriminator as mentioned here.
Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.  The function model_loss is implemented correctly.	
The function model_loss is implemented correctly.	The function generator is implemented correctly.
	Simply follow the same tips of using dropouts and weight initialisation as that for the discriminator.
	The function model_loss is implemented correctly.

#### Tip

- To prevent discriminator from being too strong, only the discriminator labels (one-sided) are reduced from 1 to 0.9. This is known as label smoothing. One can do this as follows: labels = tf.ones\_like(tensor) \* (1 - smooth)
- More Tips: refer GAN Hacks

The function model\_opt is implemented correctly.

### **Neural Network Training**

The function train is implemented correctly.

- It should build the model using model\_inputs , model\_loss , and model\_opt .
- It should show output of the  $\ensuremath{\boxed{\hspace{-0.5cm}}}$  generator  $\ensuremath{\boxed{\hspace{-0.5cm}}}$  using the  $\ensuremath{\boxed{\hspace{-0.5cm}}}$  show\_generator\_output  $\ensuremath{\boxed{\hspace{-0.5cm}}}$  function

Great work combining all the parts (functions) together and making it a Deep Convolution Generative Adversarial Network.

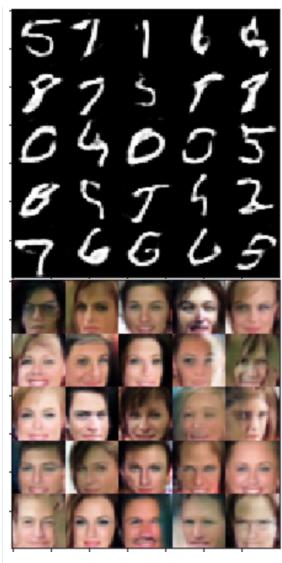
#### The parameters are set reasonable numbers.

Your choice of hyper-parameters is good. However, I suggest you to go through the Radford model in the original DCGAN paper and then choose your hyper-parameters. Anyway, I am putting forward a few of my suggestions:

- Batch Size: Try reducing batch\_size to (~16 to 32) because
  - If you choose a batch size too small then the gradients will become more unstable and would need to reduce the learning rate. So batch size and learning rate are linked.
  - Also if one use a batch size too big then the gradients will become less noisy but it will take longer to converge.
- Learning Rate: The current rate is a bit high. The DCGAN with this architectural structure remains stable with Ir between 0.0001 and 0.0008
- An important point to note is, batch size and learning rate are linked. If the batch size is too small then the gradients will become more unstable and would need to reduce the learning rate.

The project generates realistic faces. It should be obvious that images generated look like faces.

Using a similar network (3 conv blocks in D and 3 deconv blocks in G) with 2 and 1 epochs respectively, I was able to get the following results:



After making the specified changes and considering the suggestions, you will be able to come up with equally good or better results.

Also, for better visualisation purposes, simply use the  $\boxed{\texttt{n\_images}} \ \ \text{value of around 25 (5x5) in} \ \boxed{\texttt{show\_generator\_output}}.$ 

**☑** RESUBMIT

DOWNLOAD PROJECT