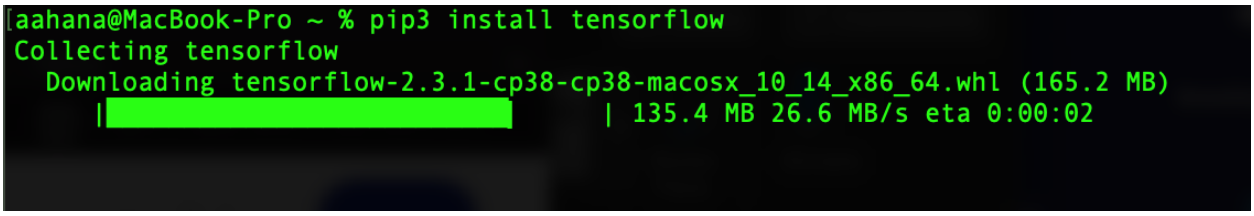
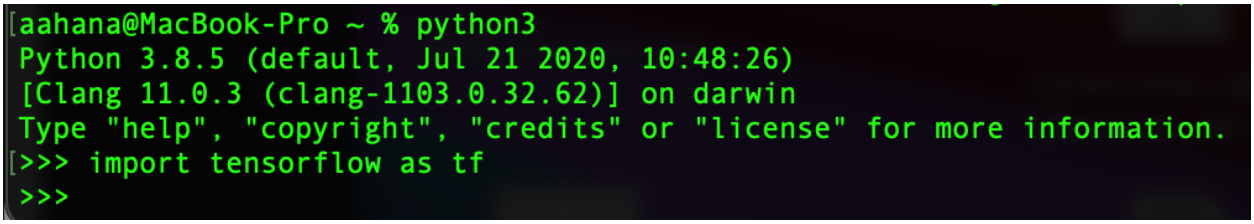
Step 1: Install tensorflow x.2

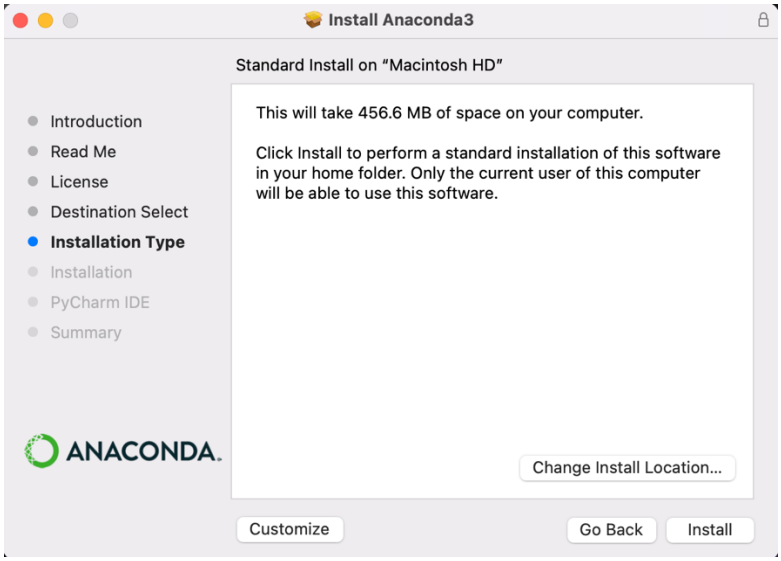
On my mac terminal window, I installed tensorflow. Screenshot is shown below:



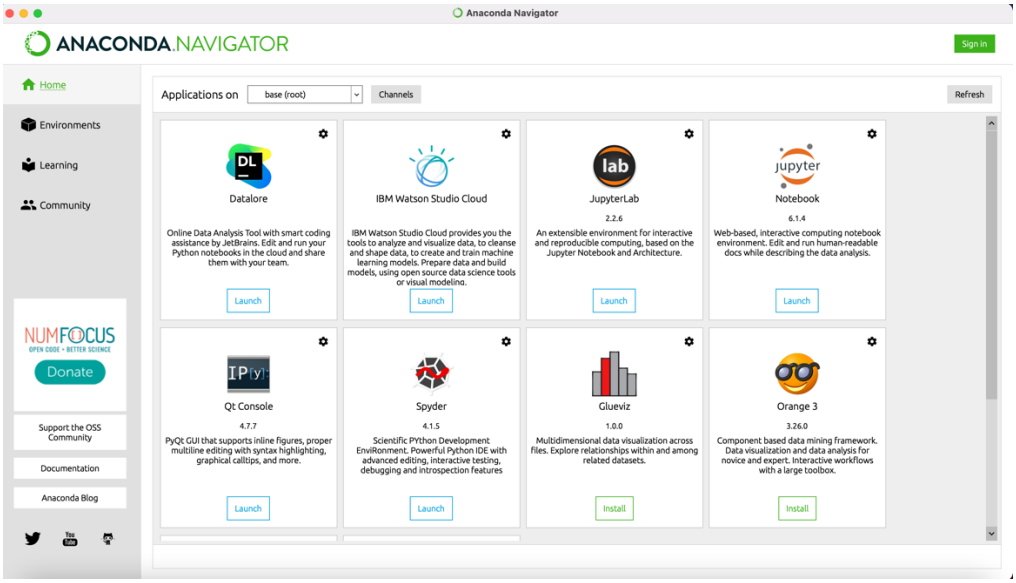
Once the installation is complete, it can be shown that tensorflow can be imported in Python:

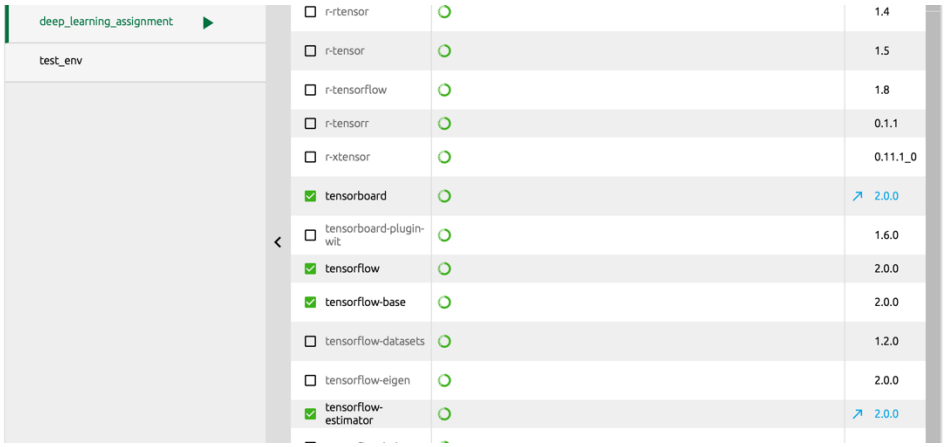


To use tensorflow with Jupyter notebook, the following steps were performed:
First, Anaconda was downloaded and installed.

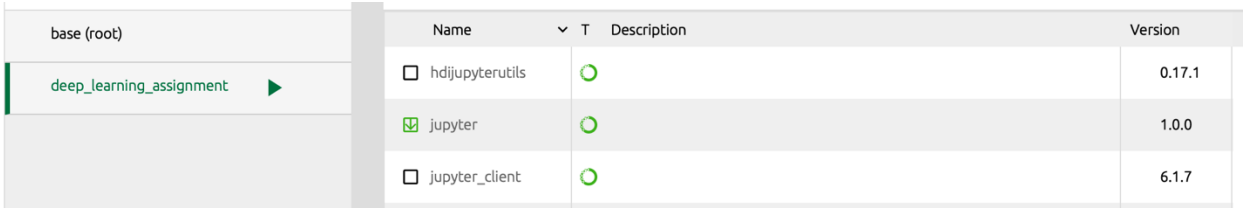


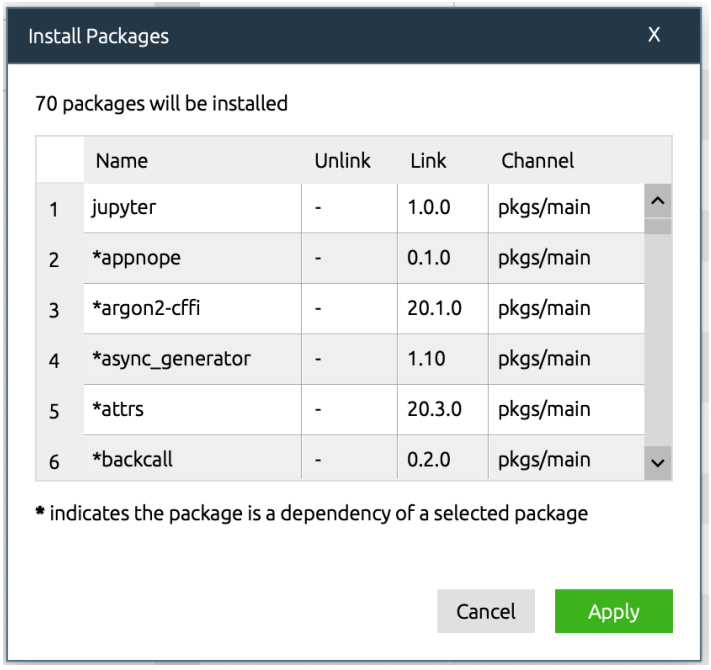
The screenshot below shows the successful installation of Anaconda. I tried to create an environment in Anaconda and add the tensorflow library to it. It did not allow it as I had Python 3.7 and tensorflow only supports up until Python 3.6. Hence, I created an environment using the command conda create -n deep_learning_assignment python=3.6. Then, to this environment tensorflow library is added.





Then I tried to open Jupyter notebook, however it was not added to the environment. So, it was added. Screenshot is shown below for this process. Once this is done, Jupyter notebook was opened and checked to see if Tensorflow is installed properly.

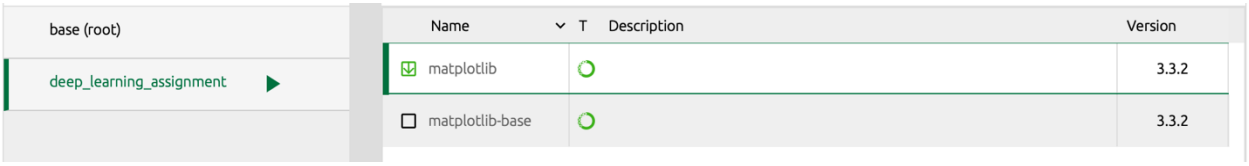





```
In [*]: import tensorflow as tf

In [ ]:
```

Step 2: Set up training for catdogmonkey

I have opened the CatDogMonkeyHomework.ipynb in my Jupyter notebook. When I run this in Jupyter notebook, I get errors for not having 'matplotlib'. Hence, I had to install it to my environment.



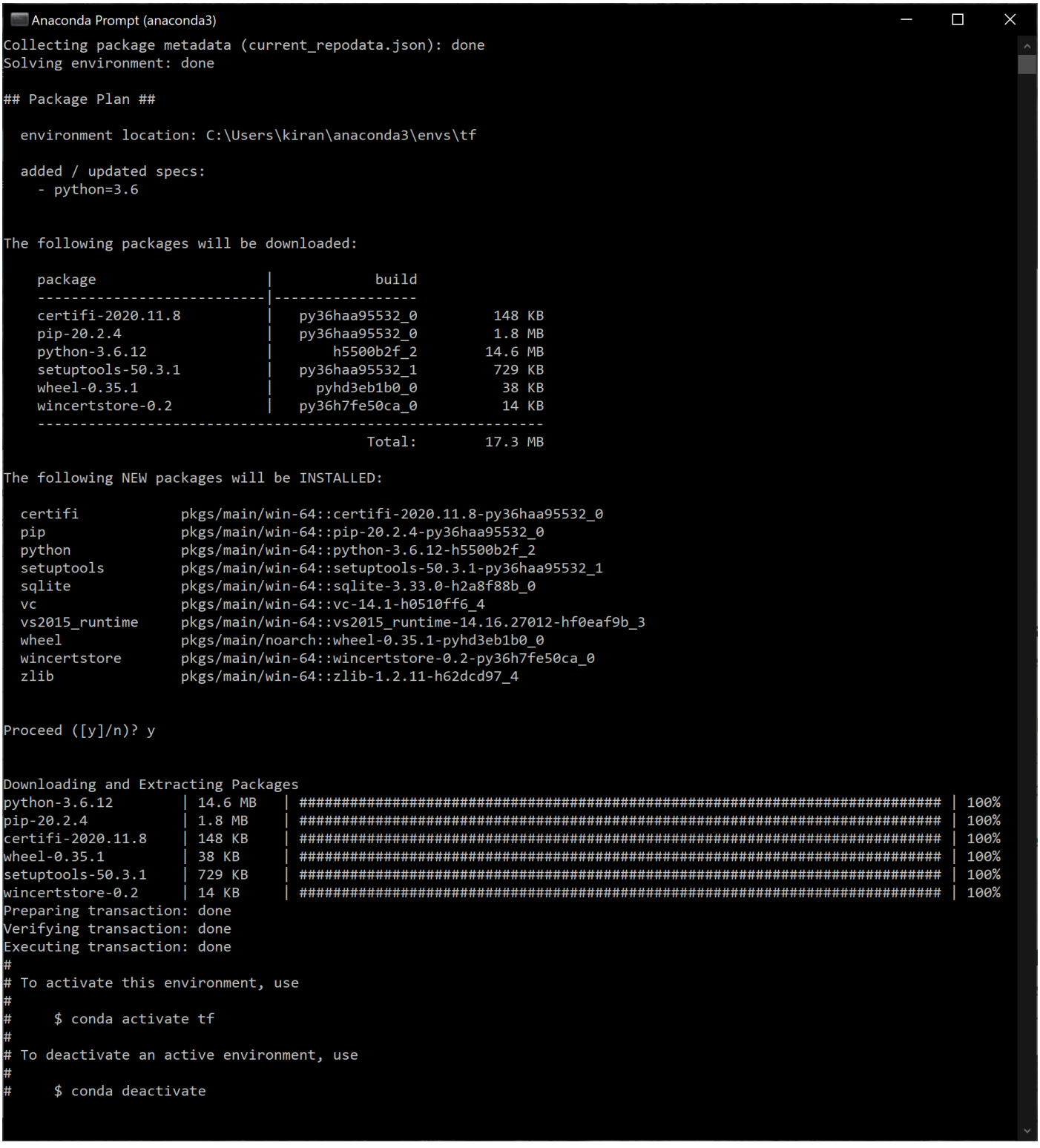
Then, I ran the cell in Jupyter again and this time, I had an error for the next section. This was a windows path error. The error is shown below:

```
In [7]: root_dir = pathlib.WindowsPath(".")
        root_temp_dir = pathlib.Path(root_dir)
        code_dir = pathlib.Path(root_dir)
        temp_dir = pathlib.Path(root_temp_dir, "_Temporary")
        temp_dir.mkdir(exist_ok = True, parents = True)

        ---------------------------------------------------------------------------
        NotImplementedError                       Traceback (most recent call last)
        <ipython-input-7-ee6079ed80c4> in <module>
        ----> 1 root_dir = pathlib.WindowsPath(".")
              2 root_temp_dir = pathlib.Path(root_dir)
              3 code_dir = pathlib.Path(root_dir)
              4 temp_dir = pathlib.Path(root_temp_dir, "_Temporary")
              5 temp_dir.mkdir(exist_ok = True, parents = True)

        ~/opt/anaconda3/envs/deep_learning_assignment/lib/python3.6/pathlib.py in __new__(cls,
        *args, **kwargs)
           1002         if not self._flavour.is_supported:
           1003             raise NotImplementedError("cannot instantiate %r on your system"
        -> 1004                                       % (cls.__name__,))
           1005         self._init()
           1006         return self

        NotImplementedError: cannot instantiate 'WindowsPath' on your system
```

The error was due to the fact that I was running my code on Mac OS (Linux/Unix filesystem). Hence to create a generic class that represents the system's path, I had to use PureWindowsPath(".") instead of WindowsPath("."). This did not fix the error and posed more issues.

Since I got a similar error with a file that I was not allowed to edit, I decided to install Anaconda and Jupyter on my virtual Windows machine the same way I did it on my MacBook. I did this using the command `conda create -n tf python=3.6`. The command prompt output looks as shown below.

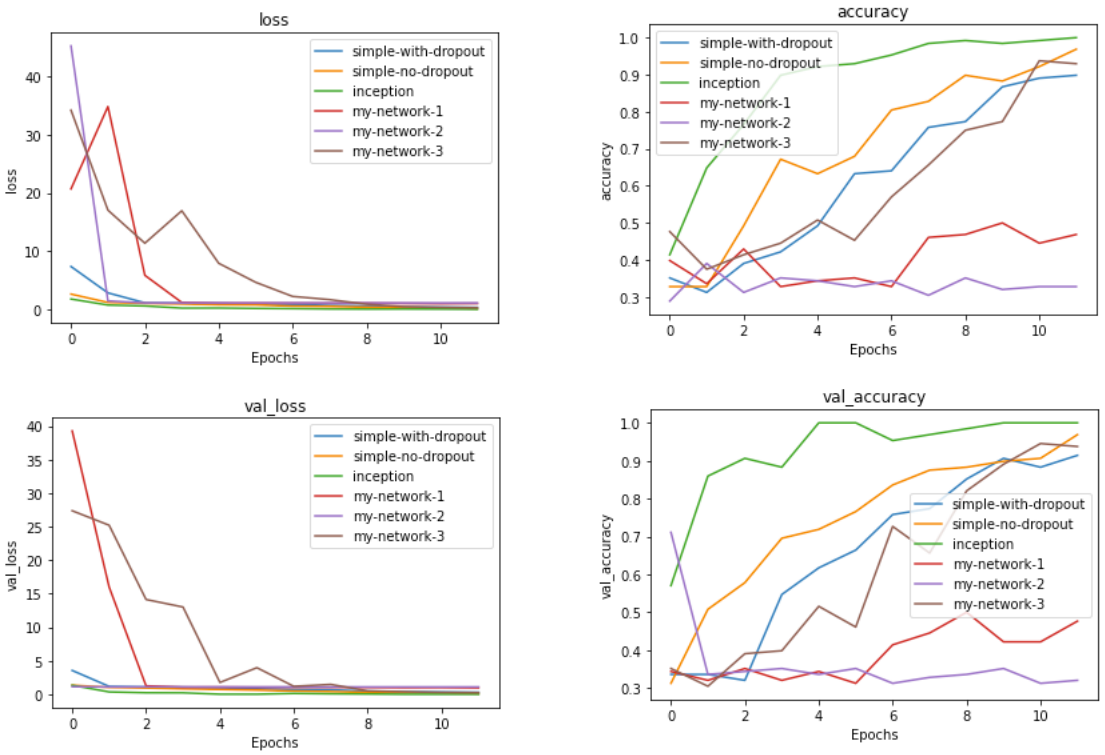
Once I had no more error, I then ran the program on my virtual Windows machine using WindowsPath() function and I did not have any more issues.



I went further to step 2 of the assignment.

Step 2: Set up training for catdogmonkey

When I run the code on Jupyter, the following output was obtained.

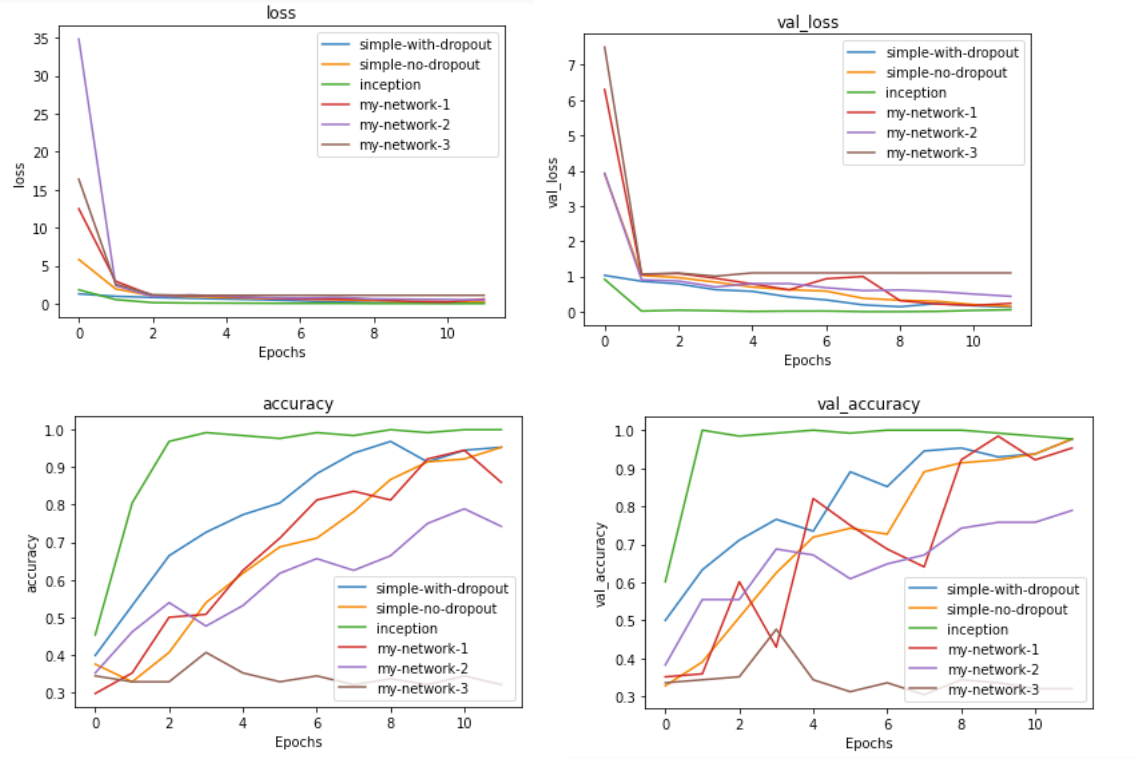


Step 3: Improve the classification accuracy

First thing I wanted to change for all three networks was number of filters. This parameter describes the number of filters that the convolutional layer will learn from. It is always recommended that the number of filters be a power of 2. Hence, from the current values = 16, I changed it to 32 (doubled it) to see if it would make any changes.

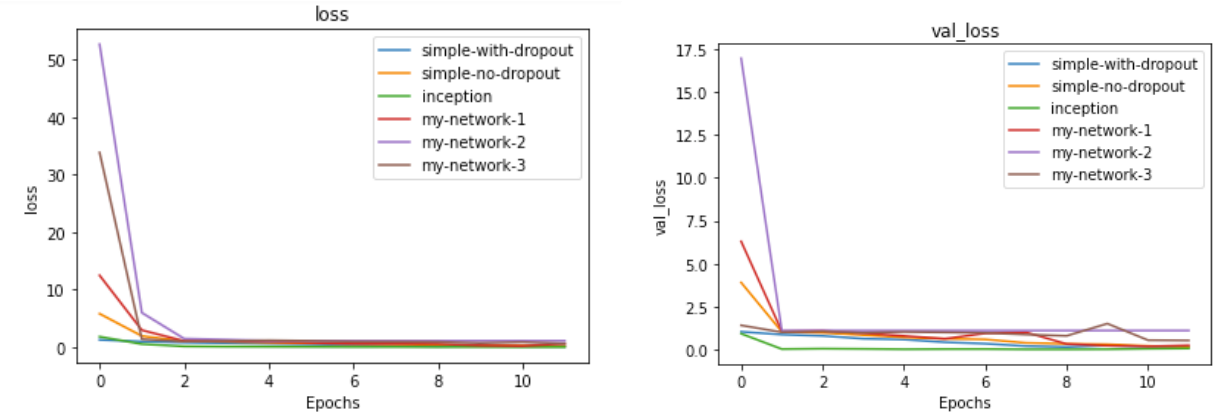
I also decided to play around with the kernel size. I decided to start by increasing the value. I changed that from 3 to 7. The code snippet is shown below.

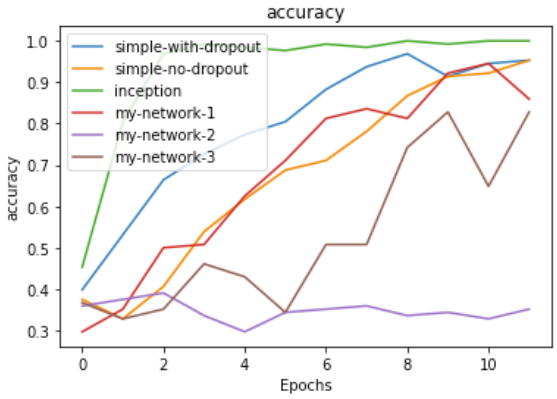
The output after both these changes looks as shown below.

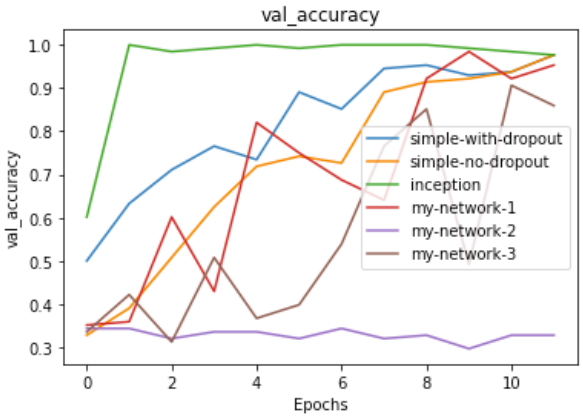


With this, I was able to definitely see a spike in inception accuracy. Both simple with and without dropouts had constant increasing accuracy. Networks 1, 2 and 3 also had pretty decent accuracy levels. However, they kept going up and down. So, I decided to play around a little bit more with the parameters to see how they would affect the outputs.

Next, I decided to increase the number of layers of network 2 and 3. Network 2 now has 64 layers and network 3 has 128 layers. The output looks as shown below. The kernel size was set to 7.

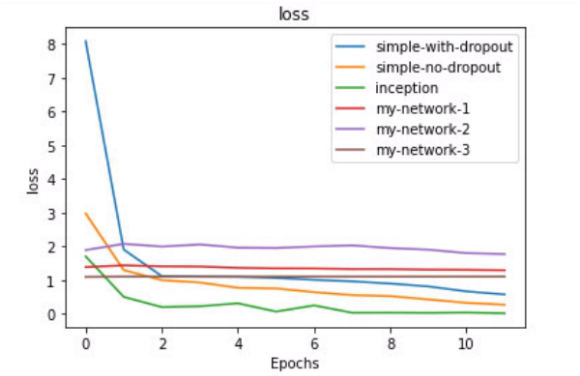


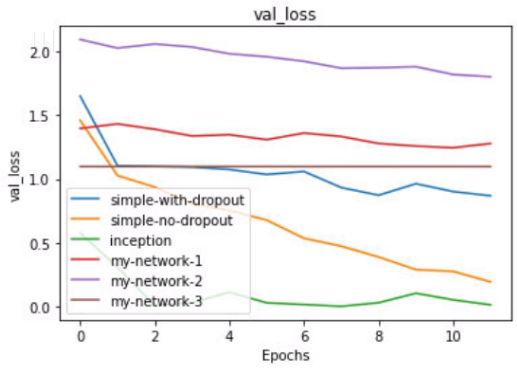


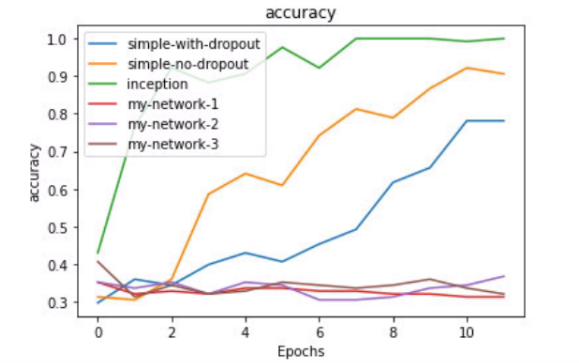


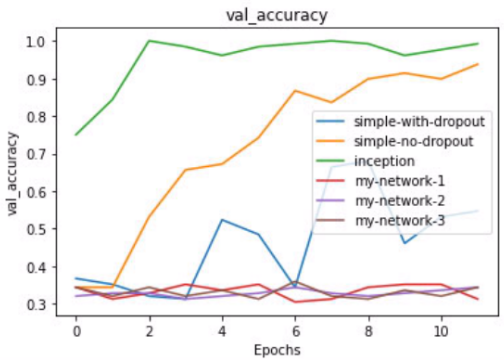
These graphs look more or less the same. Just the main difference that I saw was that network 1 has extreme ups and downs with regards to the validation accuracy values. And unlike the previous try, network 3 ended up with a better accuracy value compared to network 2, both with the training data and validation (test) data.

Next, I decide to change the activation function to see if that would make any difference. I changed the activation function of all three networks to sigmoid (I could also change it to SoftMax and try it out). I set the rest of the parameters the same. Networks 1, 2 and 3 have 32, 64 and 128 layers respectively. They all have a kernel size of 7. The output looked as follows:

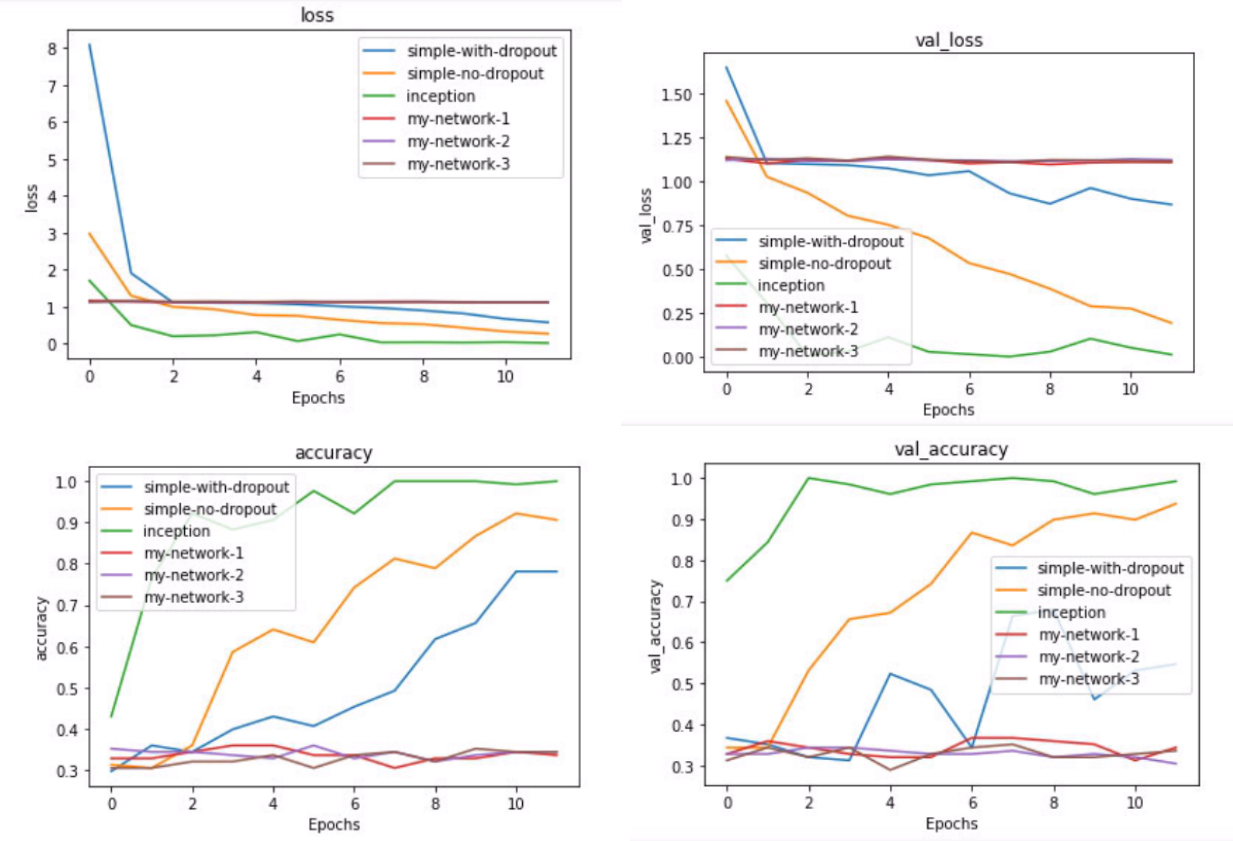








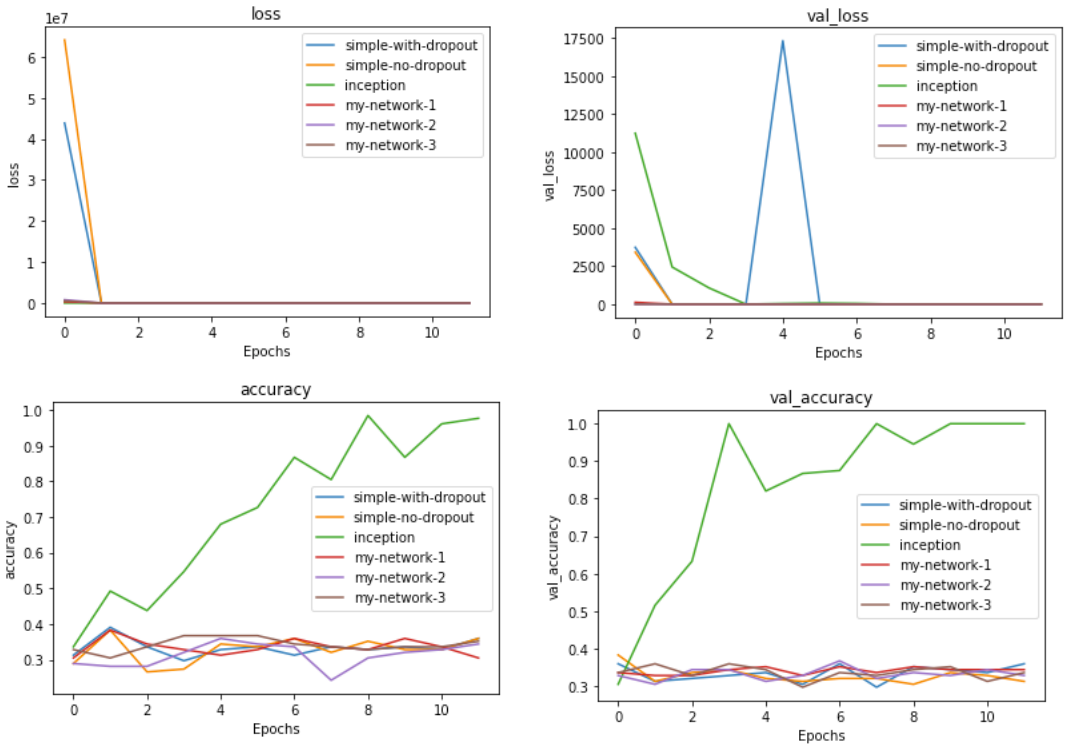
There's higher loss when the activation function is set to Sigmoid as Sigmoid activation function is used only when classification is binary, and our dataset does not involve a binary classification. Hence from the graphs it can be seen that loss values have increased, and the accuracy has gone down. So, I decided to try to set activation function as softmax, to see if the output improves.



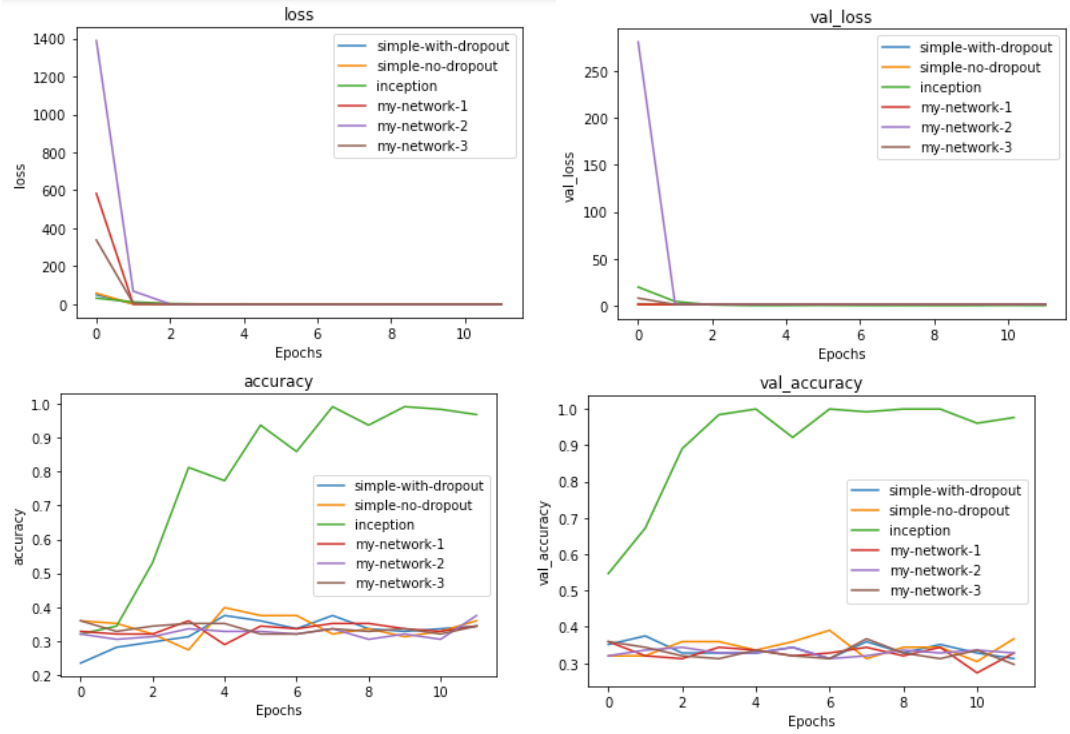
As seen on the graphs, the loss somewhat seems better when the activation function is set to softmax as compared to when it is set to sigmoid. Softmax activation function is used when the classification is a multiclass classification. However, it is still not better than when it is But, the accuracy is still not any better. I probably will come back later and set the final layer's activation function to be softmax.

So, at this point, I decided to switch gears and set the activation function back to relu. I also decided to set the number of layers back to 32 for all networks as I found that the accuracy was better in that case as supposed to having it as 32, 64 and 128 layers for network 1, 2 and 3 respectively. The kernel size was set to 7.

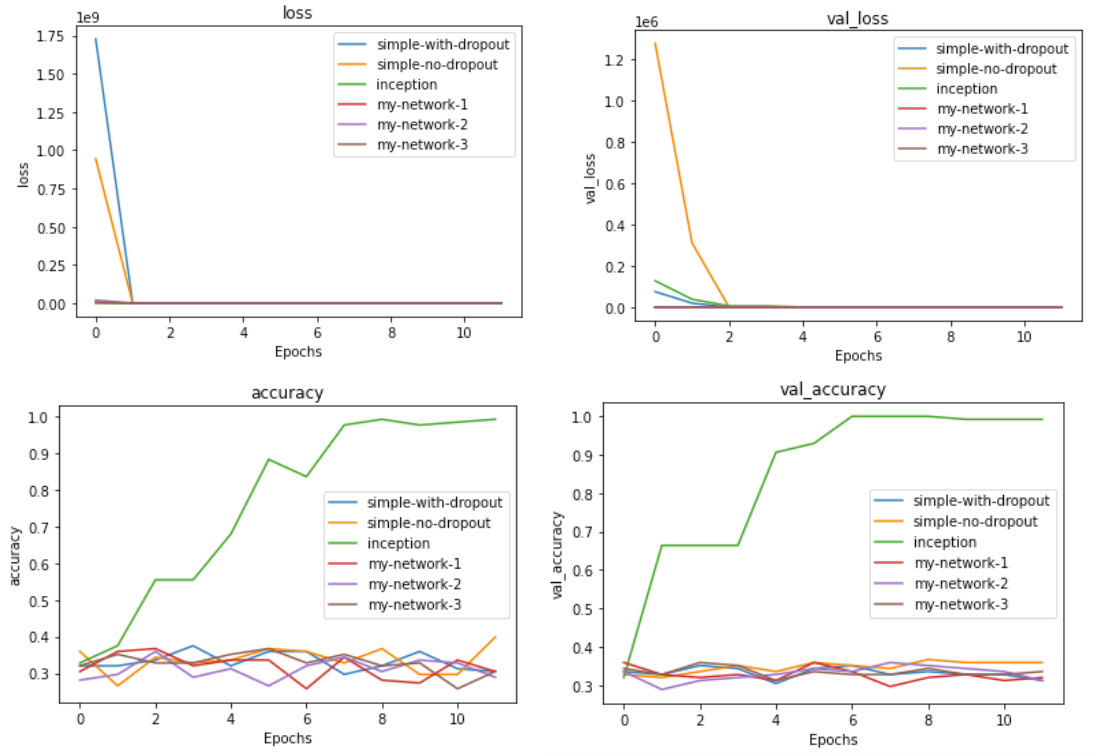
Next, I decided to explore the effects of learning rate on the model. I did this by setting learning rate using `model.optimizer.lr.assign(0.3)`. The output with this learning rate is shown below.



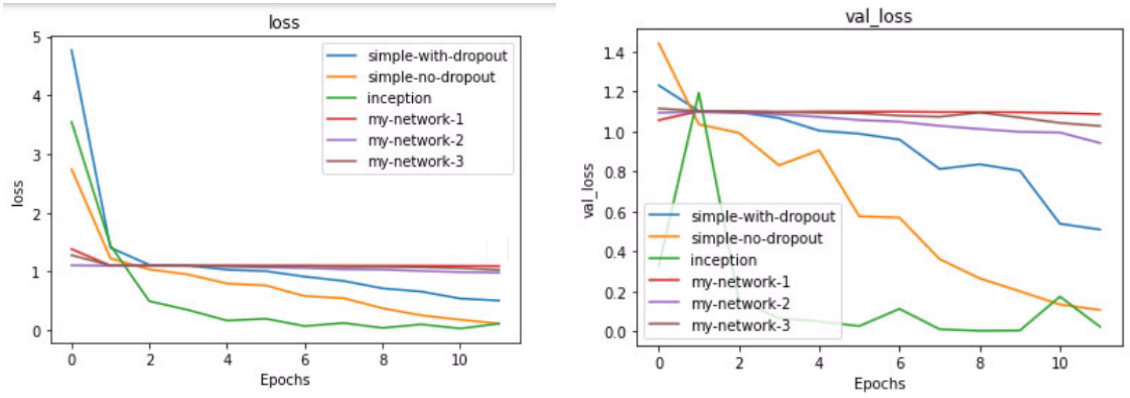
Clearly, this is not a good output. Inception is the only one that has somewhat of a good accuracy. The rest of them have their accuracy dropped. Hence, I decided to reduce the learning rate to 0.01 and check the output. Output is shown below for this setup:

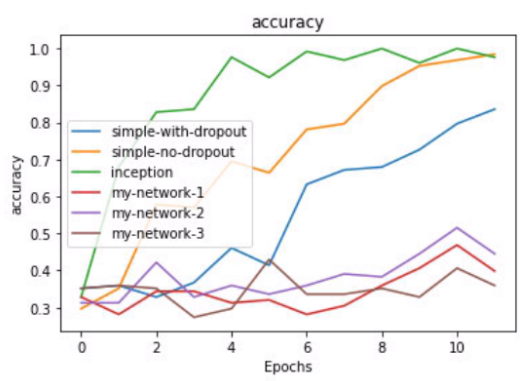


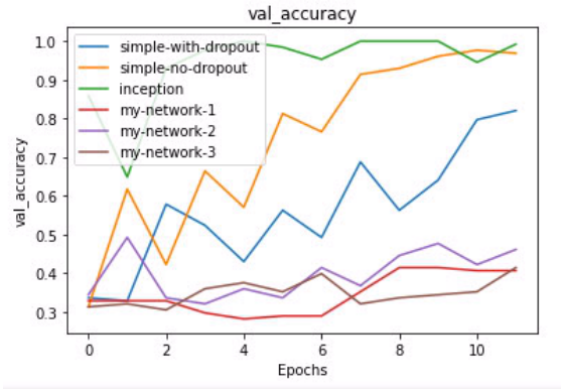
Once again, with a lower learning rate the accuracy of all networks was low except for the inception. I decided to do one last thing with the learning rate. Which is to increase its value high enough to see the difference it makes with the output. This time, I set the learning rate to 0.9. Output is as seen below.



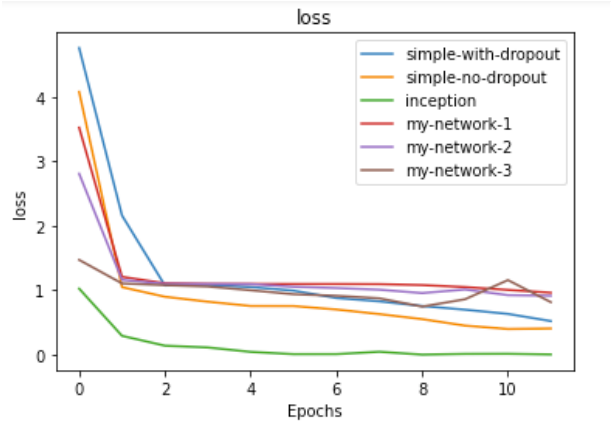
The val_accuracy is still more or less the same with a higher learning rate. Hence, I decided to not look into the number of layers in my network to see if that would help. Each network has been added with 10 dense layers after the input layer. Each of these 10 dense layers still have 10 nodes/neurons at each layer. Each of these layers still have the activation function as relu. The output is as shown below:

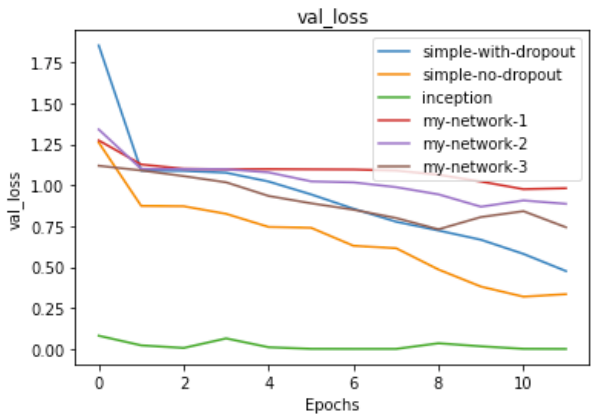


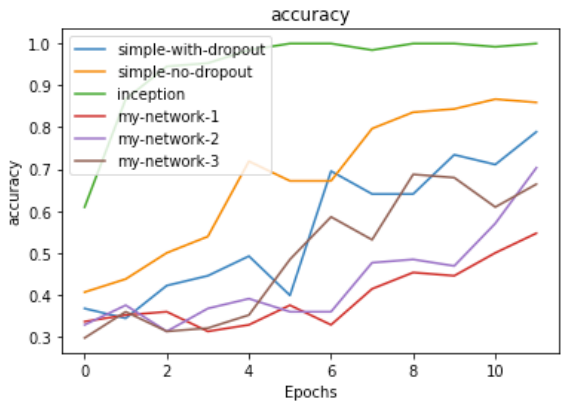


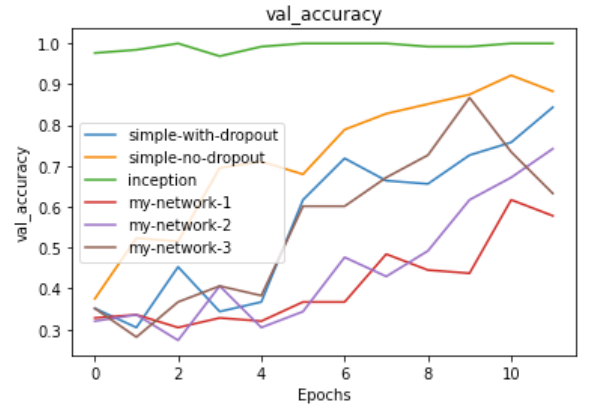


These graphs seem much better than previous trails. So, I decided to tweak a few more things around the same setup to see if it improves the output. I decided to increase the number of nodes/neurons in each layer to 50 from 10. The outputs look as shown below:



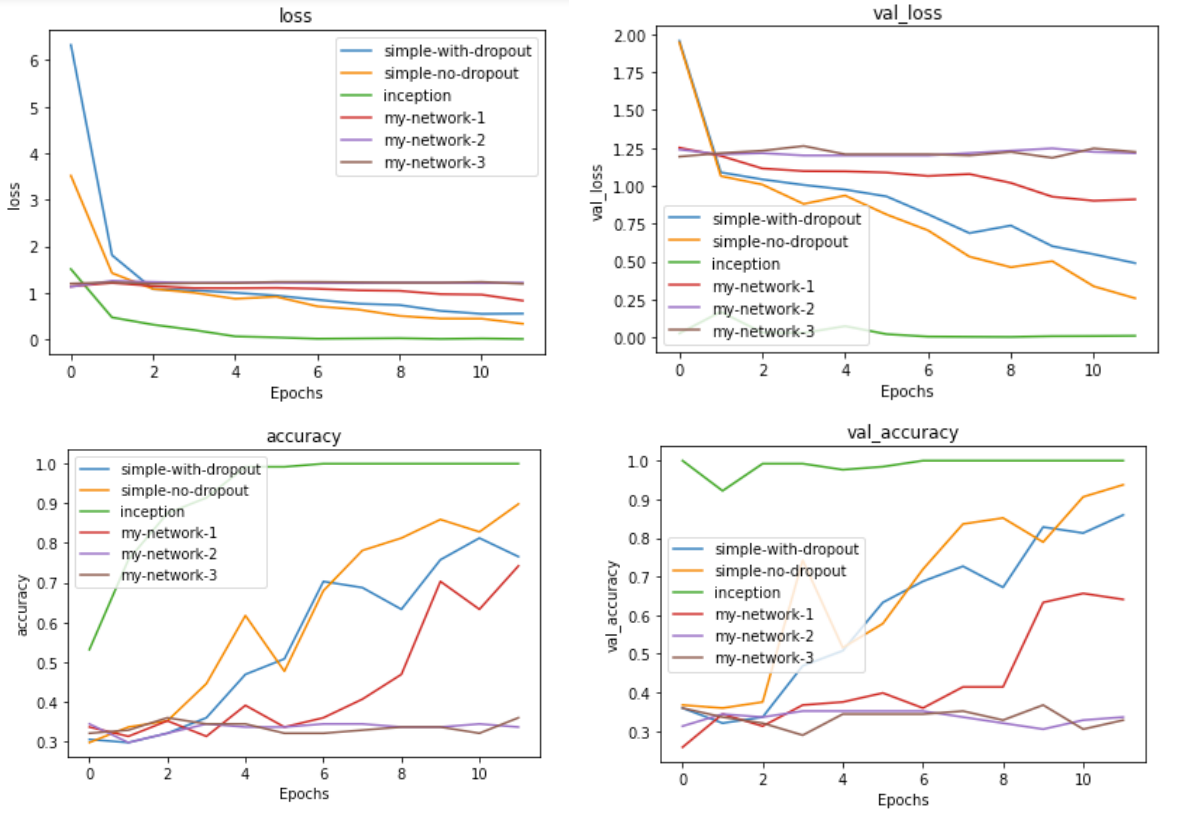




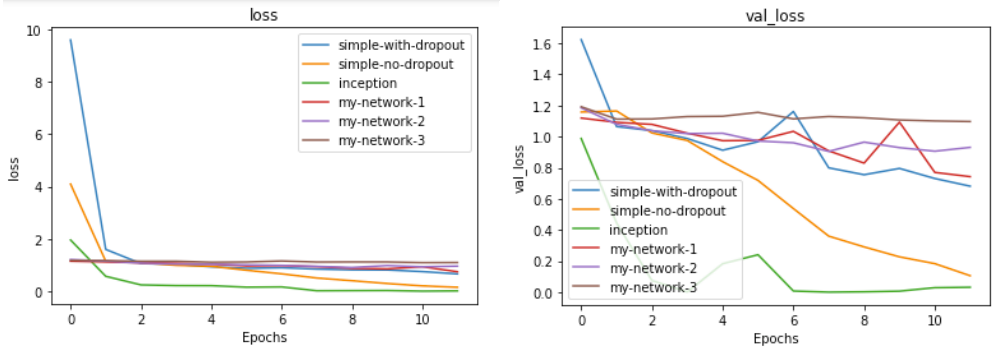


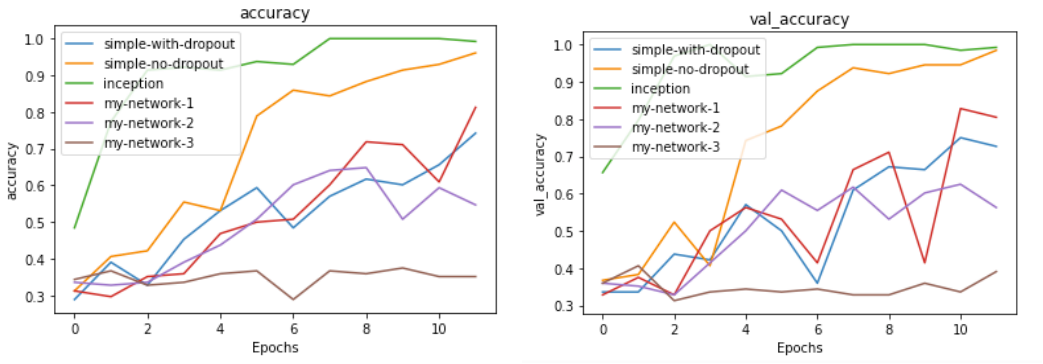
These graphs look good. There is a trend that I notice in them. I see that the loss functions are reducing (they have downward trend) as the number of epochs are increasing. Similarly, the accuracy graphs show an increasing trend as the number of epochs are increasing. In my experimentations, I am taking 12 epochs.

For the next trail, I decided to add a final layer to all networks with 3 neurons/nodes in them. I did this because we have 3 labels/classes in this classification problem. We want to classify an image as either a cat, or a dog or a monkey. I also decided to give the activation function of this final layer as softmax, since we want a probabilistic classification on the final layer to decide if it is a cat, dog or a monkey. I also made the number of nodes in the dense layer to be 100. The output is as shown below:

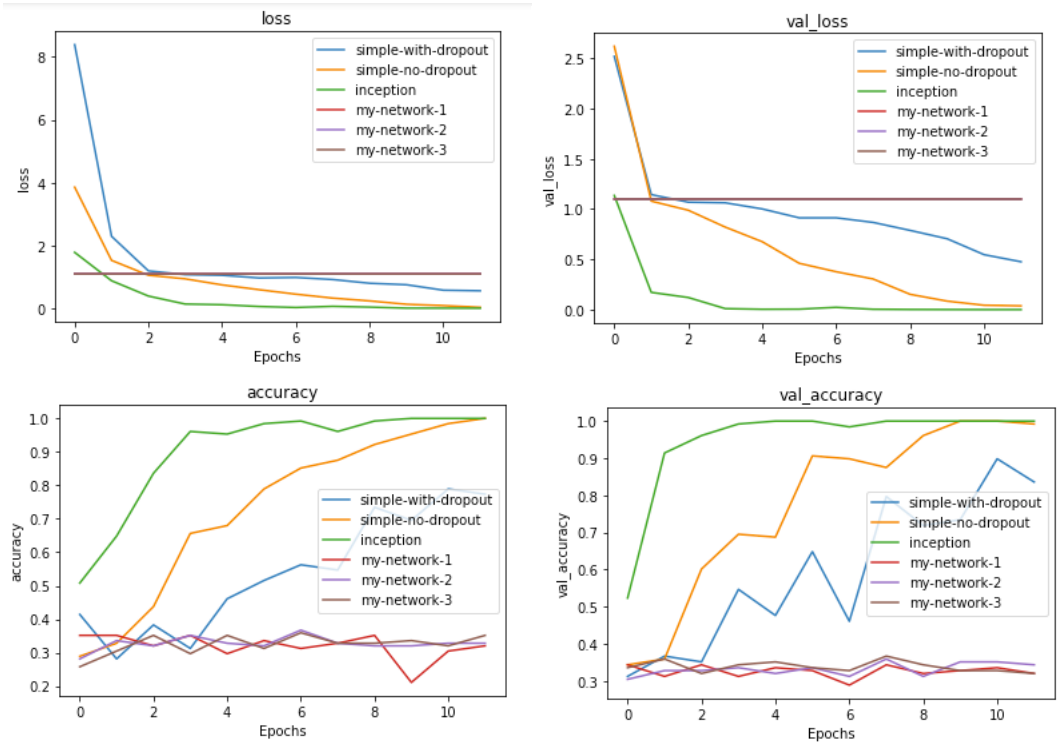


I did not think that increasing the number of nodes to a 100 in the dense layers made any significant difference. Hence, I decided to retain the final softmax layer with three nodes and to change the number of nodes in the dense layers back to 50 and get the outputs. The output graphs look as shown below:



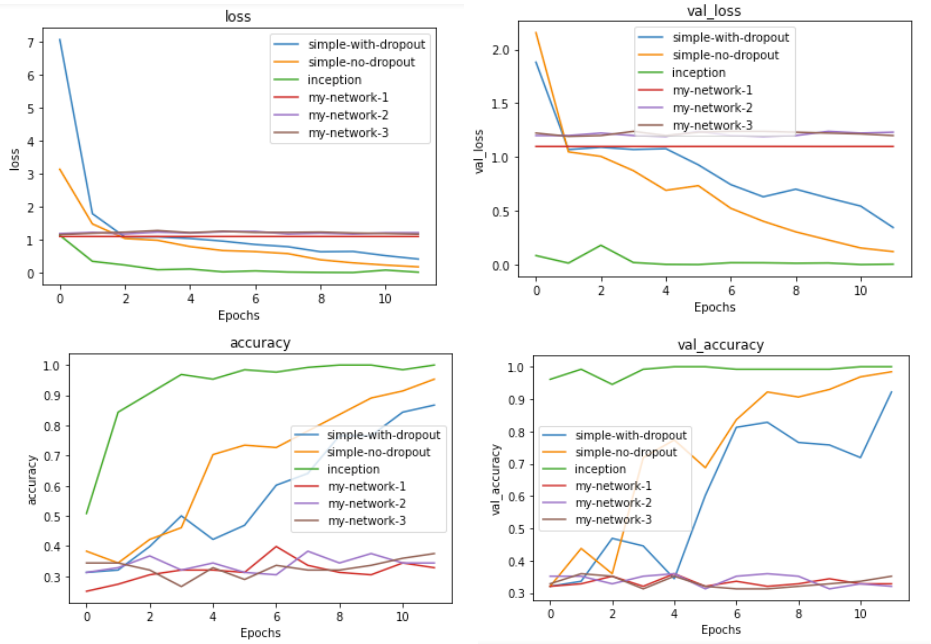


I then increased the number of dense layers to 100 in all networks. Each layer has 50 neurons. The output looks as shown below:



The accuracy levels seemed much better when the number of layers were set to 10 for all three networks. So, I decided to put them back to 10 layers. The difference that I noticed with adding 100 layers in each network is that the accuracy of simple with no dropout was 100% at the 12th epoch. The accuracy of all networks was around 35%. Inception had an accuracy of 100%.

I decided to add some dropout to the networks in order to prevent overfitting. A dropout of 30% was added. The output looks as shown below:



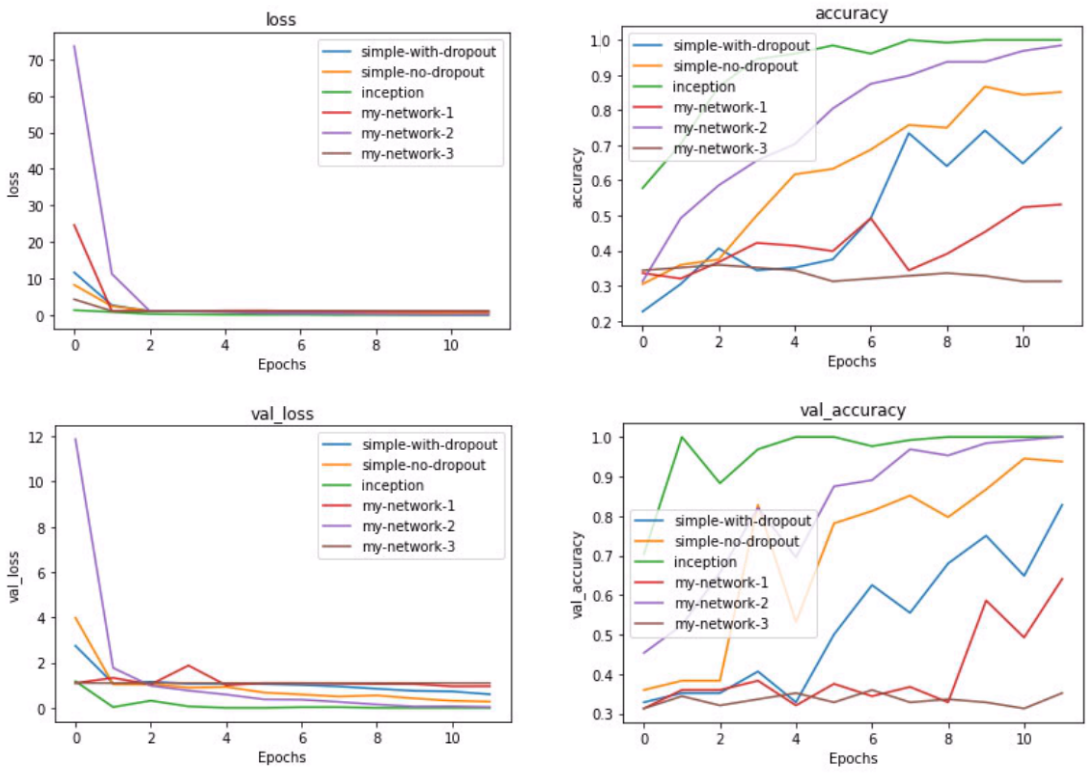
The code at this point looks as shown below:

In [4]:
```
def create_my_network_1(cfg):
    """Replace this model with an alternative model"""
    model = Sequential()
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Flatten())
    model.add(Dropout(0.3))
    for layer in range(0,100):
        model.add(Dense(50, activation='relu'))
    model.add(Dropout(0.3))
    model.add(Dense(len(cfg.CLASS_NAMES)))
    model.add(Dense(3, activation='softmax'))
    return model
```

In [5]:
```
def create_my_network_2(cfg):
    """Replace this model with an alternative model"""
    model = Sequential()
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Flatten())
    model.add(Dense(len(cfg.CLASS_NAMES)))
    model.add(Dropout(0.3))
    model.add(Dense(3, activation='softmax'))
    return model
```

In [6]:
```
def create_my_network_3(cfg):
    """Replace this model with an alternative model"""
    model = Sequential()
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Conv2D(64, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Conv2D(64, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Flatten())
    model.add(Dense(len(cfg.CLASS_NAMES)))
    model.add(Dropout(0.3))
    model.add(Dense(3, activation='softmax'))
    return model
```

At this stage, I removed the dropout since it did not increase the accuracy. I also removed the extra layers that were added and also reduced the number of nodes in all three networks. The output and the code for all three networks look as follows:



In [4]:
```
def create_my_network_1(cfg):
    """Replace this model with an alternative model"""
    model = Sequential()
    model.add(Conv2D(16, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Flatten())
    model.add(Dense(20, activation='relu'))
#   model.add(Dropout(0.3))
    model.add(Dense(len(cfg.CLASS_NAMES)))
#   model.add(Dense(3, activation='softmax'))
    return model
```

In [5]:
```
def create_my_network_2(cfg):
    """Replace this model with an alternative model"""
    model = Sequential()
    model.add(Conv2D(32, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Flatten())
    model.add(Dense(25, activation='relu'))
#   model.add(Dropout(0.3))
    model.add(Dense(len(cfg.CLASS_NAMES)))
#   model.add(Dense(3, activation='softmax'))
    return model
```

In [6]:
```
def create_my_network_3(cfg):
    """Replace this model with an alternative model"""
    model = Sequential()
    model.add(Conv2D(16, 7, padding='same', activation='relu', input_shape=(cfg.IMG_HEIGHT, cfg.IMG_WIDTH ,3)));
    model.add(Flatten())
    model.add(Dense(10, activation='relu'))
#   model.add(Dropout(0.3))
    model.add(Dense(len(cfg.CLASS_NAMES)))
#   model.add(Dense(3, activation='softmax'))
    return model
```

The final validation accuracy that I was able to arrive at was: Simple with dropout: 82.81%, simple no dropout: 93.75%, Inception: 100%, my network 1: 64.06%, my network 2: 100%, my network 3: 35.16%

GitHub Link: Deep Learning
https://github.com/monicabernard/CAP5636_Advanced_AI.git