Methods and Lessons Learned building an Image Classification Model

I experienced numerous highs, lows, and uglies in my first deep learning challenge for the Dog Breed Identification Kaggle competition. While there was no monetary prize, the incentive of climbing up the leaderboard, in if by two or so points, made the pursuit very addictive. Over the course of four weeks learning about convolutional neural networks, image classification algorithms, and manipulation of code in Keras, I was able to make it to a position of XYZ on the public leaderboard at the end of the competition. However, the reward wasn't so much the position (although, hey, top 10 is envious for anyone!), as it was truly the journey. In my post I'll share all the various lessons learned to make it to this spot and consider some next steps for how to generate a more accurate classifier.

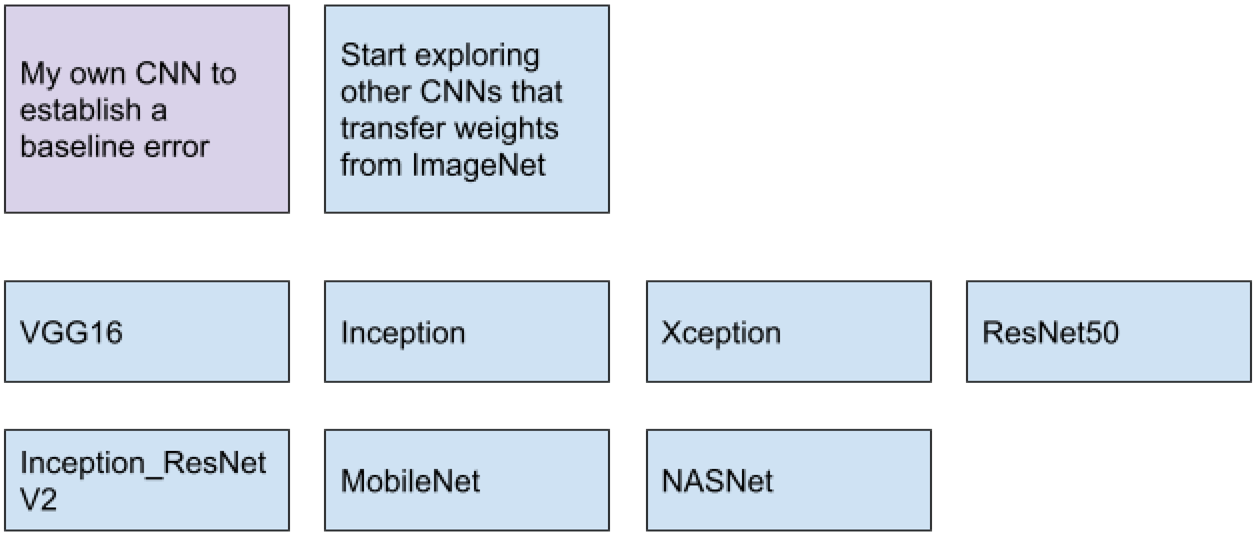
In the Dog Breeds Identification challenge, the goal is to predict the breed of a dog by any of 120 different breeds. This is truly a multi-class classification problem!

Step 1 - Consider preparing the data in two different ways: 1) Using the built in Keras Generator function, and 2) Using scikit-learn to split the data into train/test.

Step 2 - In order to preprocess the data to ready it for the Keras Generator function, I wrote a preprocessing script that sorts all the training images by each of its 120 breeds in respective 120 file folders.

Step 3 - Build my own CNN to establish a baseline error. For this one, I used the Keras Generator to fit the model on the train_generator dataset which is 70% of the original training dataset. I validated the model on the valid_generator dataset which was the 30% remaining of the original training dataset. The data was shuffled prior to applying my preprocessing script to sort them into their respective 120 subdirectory folders.

Step 4 - Extract bottleneck features from transfer learning algorithms available in Keras using weights from ImageNet. The second way of partitioning data was used for this step. In that case, I performed a random shuffle and partitioned 80% of the original training dataset into its own training set, and 20% into the validation set. Each algorithm is its own model. Fit the training set features on a Logistic Regression classifier from scikit-learn and validate on the validation features. Predict on the Kaggle Test Set for each of these models.


My own CNN to establish a baseline error
Start exploring other CNNs that transfer weights from ImageNet
VGG16
Inception
Xception
ResNet50
Inception_ResNet V2
MobileNet
NASNet

Step 5 - Develop a “soft” voting ensemble model for those models that have the lowest error from the Logistic Regression model. A soft voting ensemble outputs probabilities for each of the 120 classes, where a “hard” voting ensemble outputs only the class predictions. I focused on the Inception and Xception models, specifically.

```
Xception bottleneck features:

clf1 = LogisticRegression()
clf2 = RandomForestClassifier()
clf3 = GaussianNB()

Weights = [3, 2, 1]
```

```
Inception bottleneck features:

clf1 = LogisticRegression()
clf2 = RandomForestClassifier()
clf3 = SVC()

Weights = [3, 1, 1]
```

At this point I had six models:

1. My own CNN
2. VGG16
3. Inception trained on Logistic Regression
4. Xception trained on Logistic Regression
5. Inception trained on a Soft Ensemble
6. Xception trained on a Soft Ensemble

Validation Set	My own CNN	VGG16	Inception LogReg	Xception LogReg	Inception Soft Ensemble	Xception Soft Ensemble

LogLoss			0.6989	0.6321	0.8174	0.8093
Acc			0.8005	0.8203	0.7992	0.8152
Kaggle Submission				0.60957		

Phase 2 - Ensembling predictions through bagging.

Step 1 - Average across the test predictions outputted by all six models using a “majority vote” method. I found after much research, that there wasn’t a lot of documentation or posts about how to determine what is the best weight. As a result, I spent a lot of time essentially guessing and iterating through a variety of weights. Here are some I tried:

{w1 = 0.15
w2 = 0.20
w3 = 6.40
w4 = 4.00
w5 = 0.50
w6 = 0.30}

Scored 0.60942.

{w1 = 0
w2 = 0
w3 = 3.40
w4 = 5.50
w5 = 0.30
w6 = 0.50}

Score 0.54946.

After a lot of trial and tribulation, I learned that the worse performing models, the custom CNN and the VGG16 models should be removed altogether since they were decreasing the score. Also, I realized probably obviously, that the ensemble can only really improve by a margin of the best performing models. So, since my best performing model was the Xception Logistic Regression model, it was time to find a way to make lower log loss models.

Phase 3: New ways to train the classifier, new ways to ensemble.

Now I really buckled in and took a lot of time to learn new techniques. I needed a more accurate model.

Step 1 - Train the best performing bottleneck features using a CNN, not a logistic regression model. This is what I learned is called a “convolutional base”. Essentially, instead of using the Logistic Regression model to fit the training set, another CNN is used for the classification. You have to “freeze” the weights so that they are not updated once you call the model. Then you can build a CNN to use those weights as inputs and train the model.

I did this for the below models that had the lowest error on the bottleneck features trained on a Logistic Regression model:

1. Xception
2. Inception
3. Inception_ResNetV2

For the 1st Try Xception model:

Validation Log Loss and Acc	[0.50087296446592144, 0.8537181992586812]
Kaggle Score	0.49805

For the 2nd Try Xception model:

Validation Log Loss and Acc	[0.47396989564234554, 0.85371819914203806]
Kaggle Score	0.46940

The summary of the Xception model using the convolutional base was:

```
_________________________________________________________________
Layer (type)              Output Shape            Param #
=================================================================
xception (Model)            (None, 2048)            20861480
_________________________________________________________________
dense_1 (Dense)              (None, 256)             524544
_________________________________________________________________
dense_2 (Dense)              (None, 256)             65792
_________________________________________________________________
dense_3 (Dense)              (None, 120)             30840
=================================================================
Total params: 21,482,656
Trainable params: 621,176
Non-trainable params: 20,861,480
```

For the Inception_ResNetV2 model:

Validation Log Loss and Acc	[0.51575378223218449, 0.86839530227702189]
Kaggle Score	

The model summary:

```
Layer (type)              Output Shape           Param #
=================================================================
inception_resnet_v2 (Model)  (None, 1536)            54336736
_________________________________________________________________
dense_1 (Dense)            (None, 256)            393472
_________________________________________________________________
dense_2 (Dense)            (None, 256)            65792
_________________________________________________________________
dense_3 (Dense)            (None, 256)            65792
_________________________________________________________________
dense_4 (Dense)            (None, 256)            65792
_________________________________________________________________
dense_5 (Dense)            (None, 120)            30840
=================================================================
Total params: 54,958,424
Trainable params: 621,688
Non-trainable params: 54,336,736
```

Step 2 - Averaged the predictions using class majority weights. I tried out a few different weights. When I submitted the first ensemble, I advanced 78 spots up.

	Xception Model 1	Xception Model 2	Inception_ResNetV2 Model	Kaggle Score
Ensemble 1	0.35	0.40	0.25	0.37596
Ensemble 2	1.35	2.50	1.25	0.38223
Ensemble 3	1.35	1.50	1.15	0.37428 So far, my best score to date bringing me to #483

Step 3 - Ensemble all the **validation class predictions** from each of the 3 models above using stacked ensembling. As a result, I had 3 base learners. I used a “leave one out” approach, where most of the time in stacked ensembling it’s recommended to use a cross-validation method to combine validation predictions. Recall the validation dataset was 30% of the original training set. The validation predictions from each of the three are used as the x_meta (training set input) into a meta learner algorithm. I tried out two possible meta learners:

1. LogisticRegression.fit(x_meta, y_validation_class)
2. XGBoost.fit(x_meta, y_validation_class)

Next, the part that was truly, truly tricky for me, and admittedly took me a long time to figure out is that you need to go **back** to those three models and predict the Kaggle Test set for each of them. Those are now what I called the “base learner predictions”.

1. LogisticRegression.predict(base_learner_predictions)
2. XGBoost.predict(base_learner_predictions)

2-25-2018 Ignore Step 3 above
That didn’t work. On the last day, I did a weighted average of 16 predictions that were ensembled based on different combinations of weights. I still haven’t figured out the best way to find optimal weights, although I think using a genetic algorithm is close.

Equal weights to all submissions submission-11 to submission-26 = 0.36383

Weights to submissions submission-11 to submission-26 = 0.35893
#try this weights:
w1 = 0.15
w2 = 0.25
w3 = 0.28
w4 = 0.30
w5 = 0.30
w6 = 0.09
w7 = 0.15
w8 = 0.45
w9 = 0.12
w10 = 0.22
w11 = 0.44
w12 = 0.18
w13 = 0.33

w14 = 0.23
w15 = 0.08
w16 = 0.28

SubmissionV4 = 0.35788 BEST SO FAR

#try this weights:
w1 = 0.17
w2 = 0.18
w3 = 0.19
w4 = 0.20
w5 = 0.15
w6 = 0.21
w7 = 0.22
w8 = 0.23
w9 = 0.24
w10 = 0.25
w11 = 0.26
w12 = 0.27
w13 = 0.28
w14 = 0.29
w15 = 0.30
w16 = 0.31

Submission V5 = .35803 (worse)
#try this weights:
w1 = 0.07
w2 = 0.18
w3 = 0.09
w4 = 0.20
w5 = 0.15
w6 = 0.21
w7 = 0.12
w8 = 0.23
w9 = 0.24
w10 = 0.25
w11 = 0.16
w12 = 0.27
w13 = 0.18
w14 = 0.29
w15 = 0.10
w16 = 0.31

Submission V6

```
#try this weights:
w1 = 0.07
w2 = 0.10
w3 = 0.12
w4 = 0.15
w5 = 0.18
w6 = 0.21
w7 = 0.25
w8 = 0.29
w9 = 0.30
w10 = 0.35
w11 = 0.36
w12 = 0.40
w13 = 0.42
w14 = 0.46
w15 = 0.51
w16 = 0.60
```