Machine Learning Engineer Nanodegree
Capstone Project Proposal

Pavlos Sakoglou
November 22, 2017

Image Classification using Deep Learning with Support Vector Machines

1 Definition

1.1 Domain Background

Convolutional Neural Networks (CNNs) are a subset of Supervised Learning class of algorithms that are very similar to regular Neural Networks and aim to find an optimal predictive model that assigns the input variable to the correct label.

In contrast to the Multilayer Perceptron Architecture (MLP) that uses fully connected network layers, a CNN does not need to provide information of the entire feature space to all hidden layer nodes, but instead it breaks the input matrix into regions and then connects each region to a single hidden node. With this regional breakdown and assignment of small local groups of features to different hidden nodes, CNNs are performing very well for image recognition tasks.

On the other hand, a Support Vector Machine classifier tries to separate the data into K classes by maximizing the distance between the differently labeled data. If the data are not linearly separable, then by using an appropriate kernel function we can map the data into a higher dimension where they happen to be linearly separable and we find the linear boundary there. Finally, we transform that linear boundary back to the original lower dimensions and we get a non-linear separator.

In this project we are going to replace the standard sigmoid activation function of the penultimate layer of the network with a linear Support Vector Machine classifier and investigate performance differences. We are going to implement the standard CNN architecture as benchmark model and see how it compares with a Deep Learning SVC so that we choose the best model to implement the final solution.

1.2 Problem Statement

With the immense usage of smart phones in developed countries, people are sharing information via various types of messenger applications in unimaginable volumes. A natural and unfortunate consequence of this is message abuse in written and visual form with written texts and images. In this project we aim to partially tackle this societal issue using deep learning with SVC.

The idea is to develop and train a smart algorithm that takes an image from a message as input and detects if the image contains nudity or not. That is, the algorithm will classify the picture as "**Nude**" or "**Safe**". The receiver of the picture will have the chance to either block or open the message; having seen the class of the newly arrived image and a warning message, thus preventing harassment and unwanted information sharing.

We view this as a supervised classification problem where we train the model with a large dataset of nude and non-nude pictures of people by providing the correct labels, and then test the model with a newly received image and learn its class.

1.3 Metrics

In order to evaluate the model's performance we will rely on three main metrics, namely Precision, F_b-Score, and Recall. Notice that this application serves as an inappropriateness filter and thus we prefer any potential errors to falsely predict an image as "**Nude**", than to falsely predict it as "**Safe**".

In other words, we want to penalize more for False Negatives, which will be described by a high Recall value. To clearly demonstrate how these evaluation metrics are eligible to determine our model's efficiency, let's take a closer look at them:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positive} + \text{False Negatives}}$$

$$F_b\text{-Score} = (1+\beta^2)\frac{\text{Precision} * \text{Recall}}{\beta^2 * \text{Precision} + \text{Recall}}$$

All metrics concern both the benchmark model and its solution, and will be used in the testing process. We will compare results between the different parameter choices and architectures, and we will pick those which maximize the Recall metric. Notice that since we value Recall more than precision here, for the F_b-Score we will choose a $\beta > 1$.

2 Analysis

2.1 Data Exploration

In order to create a nude detector application we need to make sure that we collect data for both "**Nude**" and "**Safe**" classes that are representative of the general population's perception of nudity and non-nudity. In addition, we want to make sure that all races are represented proportionally in our dataset, thus our data must include people with different skin color and gender, namely white, black, and brown males and females. Notice that the skin color is supposed to generalize over races with similar skin color. For example, we assume that a person from India and a person from Middle East both belong in the same skin color category as a person from Cuba or Puerto Rico. The dataset for each class that we assemble will consist of about six thousand pictures overall.

In regards to the "**Nude**" dataset, we add one extra dimension: **Porn**, so that our model can look into non-standard nude picture patterns. The **Porn** dimension is also discretized in subcategories to make sure that is, too, consistent with respect with our data assumptions. It generalizes of three dominant porn categories: facial, group sex, and positions.

As a result, the feature space of "**Nude**" dataset consists of 6,006 pictures and the following features for both males and females. We also made sure that non of these features will dominate the other ones, thus as per each features' *importance*, we assembled proportionally many data for each one:

1. **Chest**: close up pictures of male and female chest of all colors. This features represents 9.32% of the dataset.

2. **Bottom**: close up pictures of male and female bottom of all colors. This features represents 7.84% of the dataset.

3. **Crotch**: close up pictures of male and female crotch area of all colors. This features represents 17.53% of the dataset.

4. **Front**: full on frontal pictures of male and female body of all colors. This features represents 23.37% of the dataset.

5. **Back**: full on back pictures of male and female body of all colors. This features represents 9.5% of the dataset.

6. **Porn**: several porn pictures between sets of males and females. This features represents 32.41% of the dataset.

NUDE DATASET							
Individuals	White Women	Black Women	Brown Women	White Men	Black Men	Brown Men	**Totals**
Chest	142	60	100	104	82	72	560
Bottom	155	77	99	57	37	46	471
Crotch	222	211	137	205	160	118	1,053
Front	298	234	193	287	195	197	1,404
Back	169	137	177	62	23	3	571
Totals	986	719	706	715	497	436	**4,059**

Porn							**Totals**
Facial	134	111	119	158	233	145	900
Group Sex	235	155	122	**NULL**	**NULL**	**NULL**	512
Positions	256	140	139	**NULL**	**NULL**	**NULL**	535
Totals	625	406	380	158	233	145	**1,947**

Total:	**6,006**

We did similar work for the "**Safe**" dataset. We want to make sure that the corresponding examples are strongly non-nude pictures for the model to generalize over them easily.

Nevertheless, in order to make our model sophisticated enough so it can discriminate against fully nude and partially nude, we would be required to employ bottleneck features for each dimension of the feature space. In fact, we would need to augment the data (rotate, shear, zoom etc.) in order to have enough for each feature to successfully train bottleneck models. Such task will only optimize the process, but the scope of this paper is mostly to investigate performance between the benchmark and the chosen model, and decide which classifies best. We can discuss additional performance boost techniques in another project.

As a result, the "**Safe**" dataset will consist of the following features for all skin colors and genders as above, proportionally assembled as per their importance, and we will assume that topless and underwear/swim suit pictures of both genders will be classified as nude.

1. **Lower Half**: close up amateur and professional pictures of lower body of males and females of all colors. It includes men and women in pants, and women in skirts and dresses. This features represents 12.41% of the dataset.
2. **Front**: frontal pictures of males and females of all colors. It includes full body and upper body professional and amateur pictures. This features represents 25.59% of the dataset.
3. **Intimate**: pictures of multiple people and families in intimate moments. It includes family dinners, couples of all colors and genders kissing, holding hands and being close to each other. This features represents 14.62% of the dataset.
4. **Faces**: face and portrait pictures of males and females of all colors and ages. This features represents 21.76% of the dataset.
5. **General**: pictures of people in different backgrounds, landscapes, animals, and objects in bright colors that do not strictly resemble the color of the skin. This features represents 25.55% of the dataset.

SAFE DATASET							
Individuals	White Women	Black Women	Brown Women	White Men	Black Men	Brown Men	**Totals**
Lower Half	139	119	152	105	129	132	776
Front	358	233	264	316	200	229	1600
Intimate	357	261	299	**NULL**	**NULL**	**NULL**	917
Faces	339	121	369	260	134	138	1361
Totals	1193	734	1084	681	463	499	**4,654**

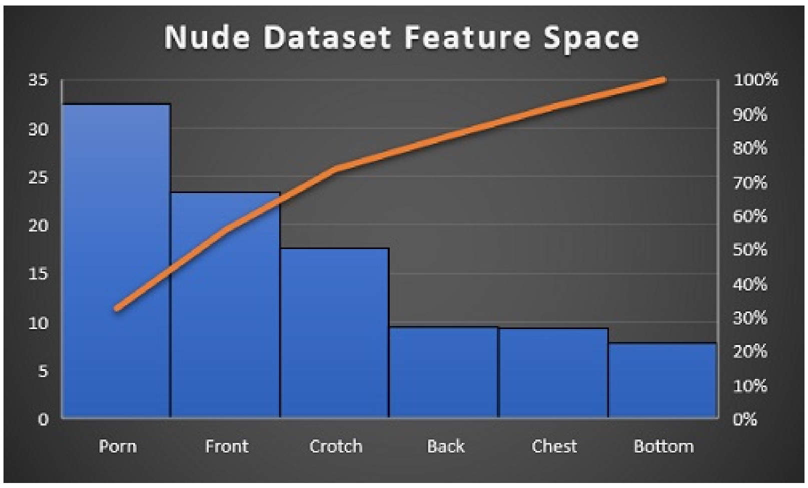
General							
People	725	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	725
Animals	201	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	201
Landscapes	98	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	98
Objects	574	**NULL**	**NULL**	**NULL**	**NULL**	**NULL**	574
Totals	1,598						**1,598**

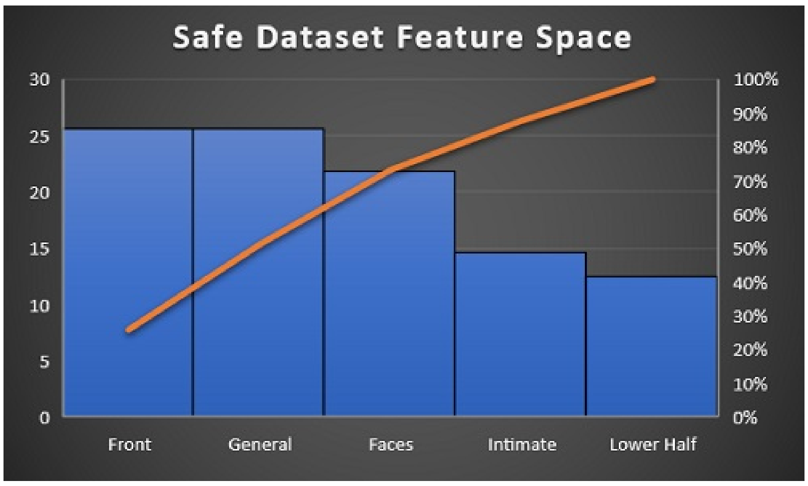
Total:		**6,252**

2.2 Exploratory Visualization

For better data comprehension, we provide the following plots that explain the feature proportionality among both datasets. Keep in mind that for the **Nude** dataset, the Porn feature implicitly includes almost all of the other features and by boosting it with more pictures proportionally is a suitable way to generalize the content of the entire class.

Fig. 2.2.1 The plots below show the data distributions for each dataset





We need to make sure that the intra-class data have high correlation, and that the inter-class data have low correlation. That way we ensure that the model will make a good job distinguishing the differences between the two underlying classes. If the inter-class data were highly correlated, then the model would have no way knowing how to separate data of different classes, and we would end up with non-intuitive, non-meaningful classification. We can visually check the correlation of the datasets by observing some sample images drawn randomly from the datasets.

Fig. 2.2.2 Sample images drawn from the Nude dataset

Fig. 2.2.3 Sample images drawn from the Safe dataset

2.3 Algorithms and Techniques

The models we are using are a Convolutional Neural Network with a Support Vector Machine classifier versus the Convolutional Neural Network with a sigmoid activation. We construct two identical models with the only difference being at the penultimate layer. We then train and test the models in the same fashion, and we compare the results. The SVM classifier model will use a linear activation function at the penultimate layer and a hinge loss function that is used in the standard SVM model. In addition we add a kernel and a bias regulizer at the final layer.

We are going to use the Sequential model from Keras framework, on which we can choose to add combinations of the following layers to define our network architecture:

1. Convolutional layers to compute the output of neurons that are connected to local regions in the input

2. Pooling layers to perform a downsampling operation along the spatial dimensions
3. Fully connected layers to compute the output of fully connected input neurons
4. Activation functions to map the dot product of the linear combination of neurons to a probability
5. Flatten operations to decrease dimensionality of layer inputs
6. Dropout operation to omit a portion of the nodes in each epoch

We will then compile the model using the aforementioned hinge loss function, which mathematically is defined as:

$$V(f(\hat{x}), y) = \max(0, 1 - yf(\hat{x}))$$

and maps the input in [0, 1] which represents a probability. Correctly classified points lying outside the margin boundaries of the support vectors are not penalized, whereas points within the margin boundaries or on the wrong side of the hyperplane are penalized in a linear fashion compared to their distance from the correct boundary.

After compiling the model, we pick values for the following training parameters in order to sufficiently train the model until there is no more room for performance improvement:

1. Number of epochs that represent full network forward and back propagation and parameter updates
2. Batch size that describes how many training sample we need to consider in each epoch

To keep track of the best parameter set and in order to use the optimal set of parameters later without having to retrain the network, we are going to use a check point file to store the optimal yet set of parameters. Thus, we will add another parameter when we fit the model, namely *callbacks=[checkpointer]*, where 'checkpointer' will the model's checkpoint file to record the optimal coefficients.

2.4 Benchmark

The main assumption of this project is that a CNN with or without a SVM classifier will perform better than random. As we saw in the data exploration part above, the two class datasets have similar size, thus it is expected to assume that an arbitrary network architecture will perform with about 50% accuracy, recall, and precision. That is, for a random network choice, there is about 50% probability for a new image to be classified as **Nude** or **Safe**.

We aim to beat this poor performance with both our networks.

3 Methodology

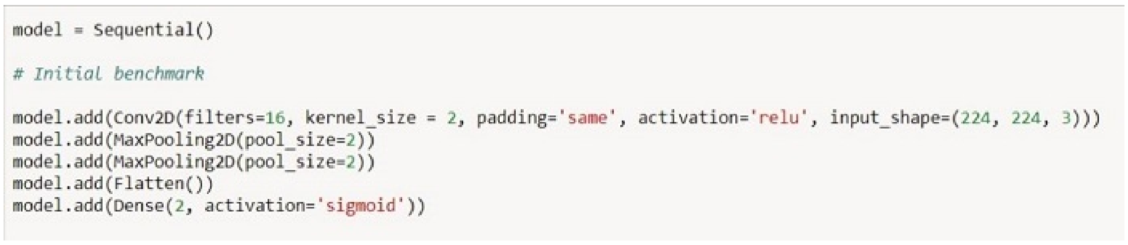
3.1 Loading and preparing the data

All the data we use in this project were collected from several websites, and as a result, many of the images are of different size and quality. In order to create a uniform dataset for both classes we will load the image paths from the local project directory, and then resize and covert the images into 224x224 RGB-valued tensors.

3.2 Implementation

The initial model is an arbitrary CNN network with several layers shown below and a sigmoid activation function at the last layer, since this is a binary classification problem. The arbitrary architecture we chose is:

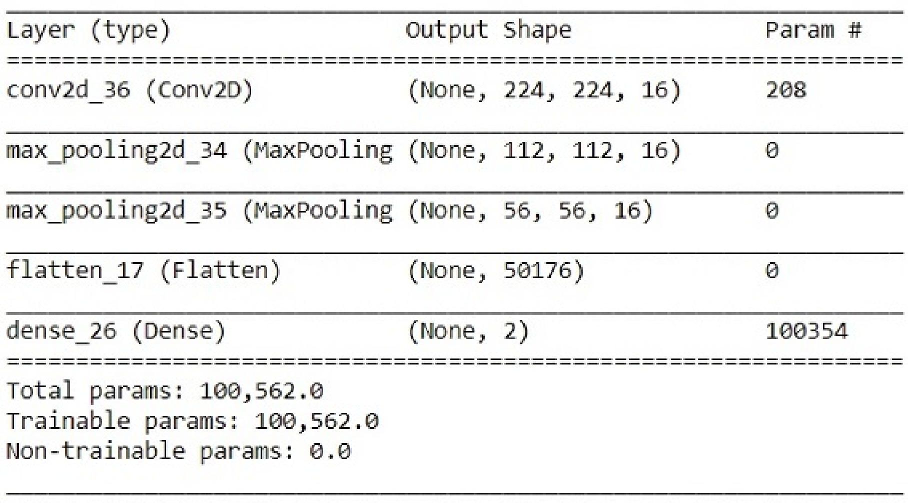
Fig. 3.2.1 Initial Model Architecture



```
model = Sequential()

# Initial benchmark

model.add(Conv2D(filters=16, kernel_size = 2, padding='same', activation='relu', input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=2))
model.add(MaxPooling2D(pool_size=2))
model.add(Flatten())
model.add(Dense(2, activation='sigmoid'))
```

having the following model summary:

Fig. 3.2.2 Initial Model Summary

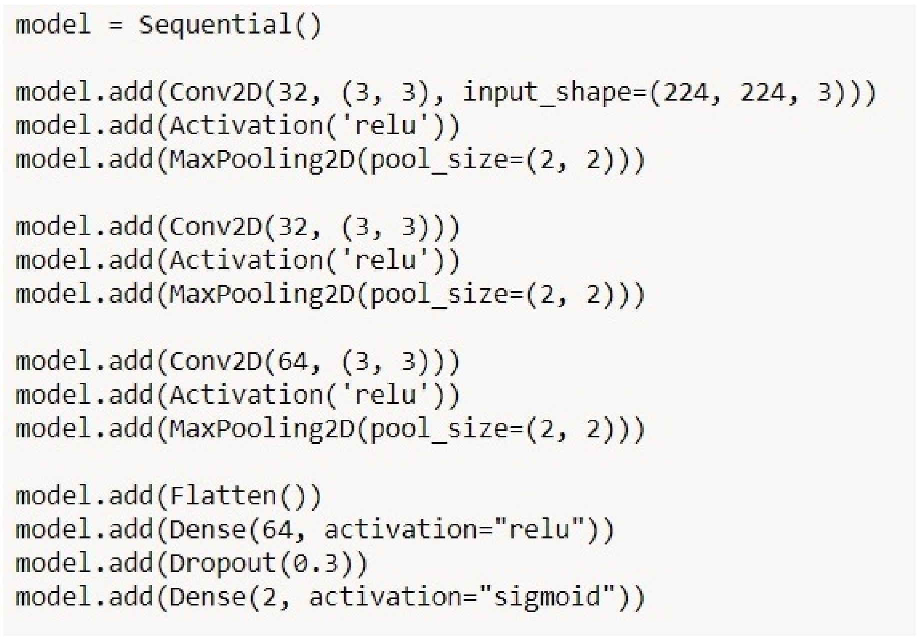


```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_36 (Conv2D)           (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 112, 112, 16)      0
_________________________________________________________________
max_pooling2d_35 (MaxPooling (None, 56, 56, 16)        0
_________________________________________________________________
flatten_17 (Flatten)         (None, 50176)             0
_________________________________________________________________
dense_26 (Dense)             (None, 2)                 100354
=================================================================
Total params: 100,562.0
Trainable params: 100,562.0
Non-trainable params: 0.0
_________________________________________________________________
```

We then compile and train the model using binary crossentropy loss function and ’rmsprop’ optimizer, 10 epochs and 25 samples per epoch (batch size) to obtain 74.6% accuracy, 77.8% precision, 70.9% recall, and 0.716 F3-score.

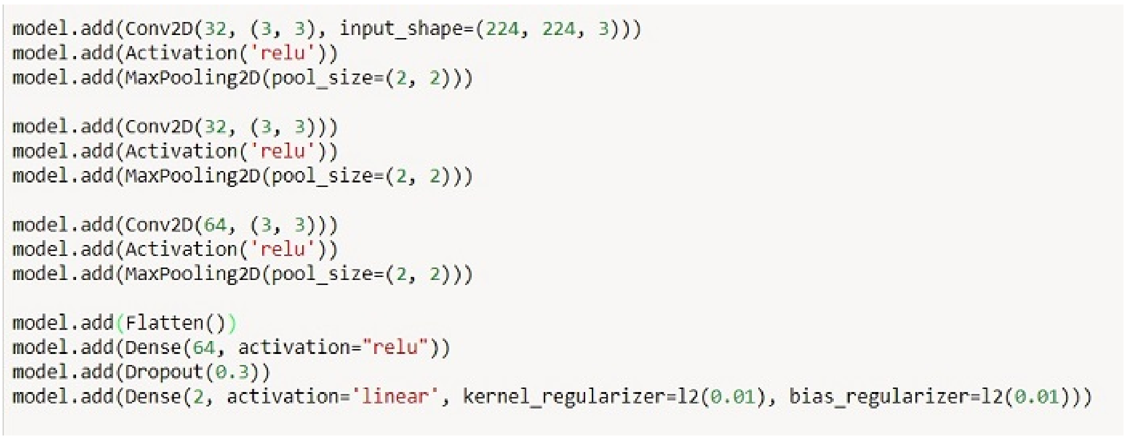
For the target models, we will choose a deeper network with a small amount of nodes in each layer. As mentioned earlier, both the CNN and the CNN-SVMC models will share the same architecture except the final layer and the loss function. Such architecture would look like:

Fig. 3.2.1 Target CNN Model Architecture



```
model = Sequential()

model.add(Conv2D(32, (3, 3), input_shape=(224, 224, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(2, activation="sigmoid"))
```

Fig. 3.2.1 Target CNN-SVMC Model Architecture



```
model.add(Conv2D(32, (3, 3), input_shape=(224, 224, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.3))
model.add(Dense(2, activation='linear', kernel_regularizer=l2(0.01), bias_regularizer=l2(0.01)))
```

When compiling the CNN model we use binary cross entropy loss function, and when compiling the CNN-SVMC model we use the hinge loss function, defined above.

As you can see, we added a dropout layer towards the end to avoid over-fitting, as well as a Flatten layer so we can flatten the input and prepare it for the final Dense layer. After some experimentation, we decided to go with the rectified linear unit activation, and three max pooling layers. This architecture gives the best results yet as we will see below.

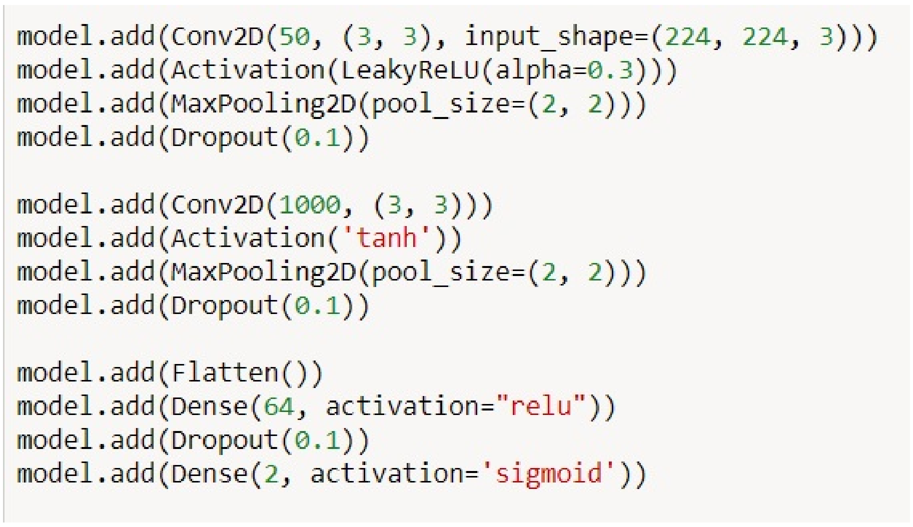
3.3 Refinement

As mentioned earlier, it took several steps to derive the above architecture. We resorted to parameter tuning and we tried several model architectures, some deeper and some more swallow. In particular, we tried to use 'tanh' and 'LeakyReLU(alpha = 0.3)' activation functions, be apparently in our example they do not perform as well as 'relu'.

In addition, we tried to add several dropout layers with a smaller dropout rate each, however it also didn't perform as well. Furthermore, we experimented with an architecture that has many convoluted layers in the middle, it drops out 10% of the nodes each time, and uses different activation functions in each layer.

That said, one of the architectures we tried was the following:

Fig. 3.2.1 Alternative Model Architecture



```
model.add(Conv2D(50, (3, 3), input_shape=(224, 224, 3)))
model.add(Activation(LeakyReLU(alpha=0.3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Conv2D(1000, (3, 3)))
model.add(Activation('tanh'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.1))

model.add(Flatten())
model.add(Dense(64, activation="relu"))
model.add(Dropout(0.1))
model.add(Dense(2, activation='sigmoid'))
```

and after some tuning we improved results by deriving the architecture of the previous section. Notice that this alternative architecture is estimating 187,076,594 parameters and the training process is extremely slow without improving the results at all.

In regards to the number of epochs and batch size, we came to conclude that a smaller batch size is preferred in order to avoid but model generalization. That is, we train the model with batch size equal to 16 training samples. The performance decreases as the batch size increased over 50 and remained the same as the batch size was ranging between 20 and 30. The number of epochs in all trials was kept between 5 and 10. In most cases, the loss of the model wasn't improving after a second or third epoch. Nevertheless, on the final testing of the models, the batch size picked is 16 and the number of epochs is 20.

4 Results

4.1 Model Evaluation and Validation

4.2 Justification

5 Conclusion

5.1 Free form visualization

5.2 Reflection

5.3 Improvement