Mostly Not Smiling to Mostly Smiling: Predicting when a Yearbook photo was taken

Amin Anvari
UT Austin

Diego Garcia-Olano
UT Austin

Farzan Memarian
UT Austin

Abstract

A century of portraits is a historical image dataset that comprises a large scale collection of yearbook portraiture from the last 120 years in the United States that Shiry Ginosar et al[3]. made public for the first time in 2015. The dataset is labeled with the year that each photo is taken. In this project, we train different deep (convolutional) models to predict the year a novel photo in the yearbook is taken. We framed the task at hand in different frameworks, such as classification, regression and a combination of both. We trained several networks and demonstrated that VGG16 architecture works the best in our task. We obtained test accuracy of 3.0% and mean L1 error of 4.5 years. At the end of this report, we demonstrate some of the visualizations that we made to get insight in to what the model is learning. The code for this project is available at https://github.com/anvaribs/cs395t-f17

1. Introduction

A paradigm shift in Computer Vision tasks has occurred in the past few years moving from systems based on hand generated features to ones where features are learned through the use of Deep Convolutional Neural Networks. The process of determining the architecture for these models, the number and types of layers they contain, their activation functions, the associated hyper-parameters, methods to avoid over fitting, and the subsequent training and fitting can be quite time and labor intensive. One of the most powerful ideas to come from deep learning is that its possible to transfer knowledge from a network that has been trained on a given task and apply that knowledge towards a separate task. This process is called transfer learning[5]. Its then beneficial to fine tune the weights of this combined network for the task at hand. This process cuts down dramatically on the amount of time needed to train a model.

In the following project, we were given a set of around 25,000 high school yearbook photos taken from between the years 1905 and 2013 in the United States which have been cropped and converted to gray scale. Our task is to predict the year a yearbook photograph was taken using only that image's pixel values. We do so by taking a neural network which is trained for image classification on the ImageNet[2] dataset and then adapt it for our task. Figure 1 shows sample images from the dataset and Figure 2 shows the distribution of the data from the original paper's dataset, our project's data has no regions and is a subset of the original data.

To this end, we use various pre-trained networks that are available in open source deep learning frameworks such as tensorflow and keras. A pre-trained network is simply a saved network previously trained on a large dataset, typically on a large-scale image classification task. If this original dataset is large enough and general enough, then the spatial feature hierarchy learned by the pre-trained network can effectively act as a generic model of our visual world, and hence its features can prove useful for other computer vision problems, even though these new problems might involve completely different classes from those of the original task.

For our task, we will use large convnets trained on ImageNet dataset (1.4 million labeled images and 1000 different classes). Since the ImageNet contains many different classes and is not specifically portrait images, we decided to fine-tune the network for our task.

We experimented with 4 different architectures. VGG16[6], InceptionV3[7], Xception [1] and ResNet[4]. VGG16 architecture, developed by Karen Simonyan and Sndrew Zisserman in 2014 proved to be the best performing architecture for our face-to-age prediction task. Although VGG16 architecture is much older compared to other high-performing models, the ease of understanding what is going on behind the scene and what network is learning enabled us to train the network much better and get the highest performance in terms of L1 error out of it.

We also concluded that the best framework to model the problem is through classification with classes corresponding to 104 years between 1905 to 2013 corresponding to the labels that we have in our training and validation datasets.

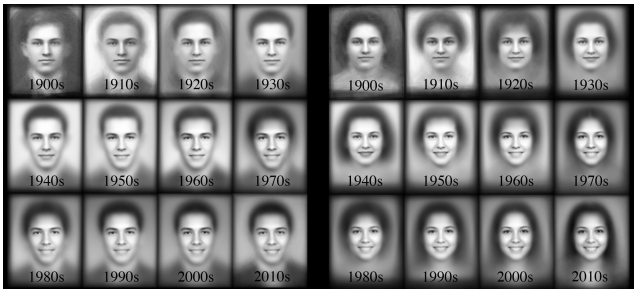


Figure 1. Yearbook Photos for Men and Women 1900-2010

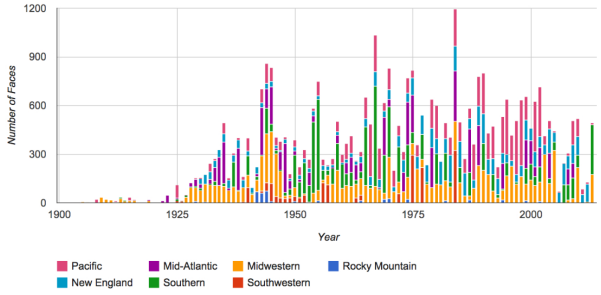


Figure 2. Distribution of portraits per year and region

2. Technical Section

The following section discusses how transfer-learning, fine-tuning, hyper parameter tuning, different optimizers, learning rates, regularization through data augmentation, batch normalization and dropout were used in the training and validation of our models.

2.1. Methods

As discussed earlier, we pose the task of predicting the year each photo is taken using the frontal-facing high school yearbook photos as a 104-way year-classification task between the years 1905 and 2015. We set aside about 5% of the data as the test set and use the remaining 95% for training and validation. Note that the distribution of photos per year is very unbalanced (figure 2).

We use the Keras implementation of the VGG, inceptionV3, Xception, and ResNet network architectures that are pretrained on the ILSVRC dataset in all our experiments. We fine-tune the final fully connected layers of the networks on the yearbook training data to predict the year at which a photo is taken. We train our networks for 40 epochs (around 15,000 steps) using ADAM optimizer (LR = 0.001) in the pre-training step and SGD with a very small learning rate in the fine-tuning step. As we expected, fine-tuning on the the yearbook dataset improves the L1 error from around 8-9 years to around 5 years.

Since the size of the training dataset was moderately large, we decided to train the network in a pre-training and fine-tuning fashion. In short, we have taken the knowledge learned from an image recognition task on ImageNet dataset and transferred it to the face-to-year prediction task. The reason this is helpful is that a lot of the low level features such as detecting edges and detecting curves, might help the learning algorithm do better in our task.

We frame the problem as a classification, regression, and combination of both to evaluate the effect of these choices. We tried different loss functions. Since the pre-trained networks are originally designed to be used for classification task, we started out by using a categorical crossentropy loss function and framed the problem as a 104-way classification task with a softmax layer at the output of the network.

Motivated by the fact that this loss function is ignoring the inherent structure of the ordinal classes, we devised some other loss functions to exploit the relationship between different classes. We experimented with a pure L1 loss function and also with a combination of categorical cross-entropy loss function and L1 loss function.

For the L1 loss function, we tried both a loss that only considers the class with maximum probability to calculate L1 loss and also a loss function that is weighted average of all the class predictions that softmax layer is outputting. This is a better loss function since it take into consideration the probability of all classes and also does not suffer from the zero gradient. Following you can see all the loss functions that we used:

$$
\begin{aligned}
loss_1 &= |year_p - year_t|^2 \\
loss_2 &= Cross\_Entropy(y_p, y_t) \\
loss_3 &= Cross\_Entropy(y_p, y_t) + C|year_p - year_t|^2 \\
loss_4 &= Cross\_Entropy(y_p, y_t) + C|year_p - year_t| \\
loss_5 &= Cross\_Entropy(y_p, y_t) + C\|y_p - y_t\|_1
\end{aligned}
$$

where C is a coefficient that determines the relative importance of classification loss versus the regression loss. y_t is the one hot encoded vector of true label for the current example and y_p is the output of softmax. $year_p$ and $year_t$ are the predicted year and true year respectively

The yearbook data was preprocessed via the same procedure as the original architecture and we additionally performed standard data augmentation of the images to help with our prediction task and to prevent overfitting. The learning rate was additionally reduced via keras' ReduceLROnPlateau callback function which monitors the model's 'val mean L1 distance' and reduces the learning rate when that value has stayed constant for a set number of epochs.

2.2. Hyperparameter tuning:

Here we describe systematically organizing hyperparameter tuning process to find a good setting for the hyperparameters. One of the difficulties in deep learning is the sheer number of parameters that the practitioner has to deal with, ranging from the learning rate LR α, the momentum term β, learning rate decay, mini-batch size, Adam: β_1, β_2, ϵ; number of layers, number of hidden layers for different layers, mini-batch size, if using Adam optimizer: Adam: β_1, β_2, ϵ The order that we tuned the hyperparameters: Learning rate $\rightarrow$ mini-batch size $\rightarrow$ hidden units number. After coming up with good values for these hyperparameters, we tried to do tuning for learning rate decay and we actually did not tune, but set standard values for the momentum term β = 0.9 and Adam parameters $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$. Due to sheer number of hyper-parameters, instead of doing a grid search, we decided to try to make a random sampling search to more richly explore the set of possible values and to make it more likely to find a value that works well. We also used a coarser to finer search scheme. So we started with a coarse random sampling first and then zoomed in and sampled more densely within that space. We tried to come up with an appropriate scale to pick hyperparameters. For the learning rate hyperparameter, we used a log scale for our grid. We experimented with $\alpha = 0.0001, \alpha = 0.001, \alpha = 0.01$ and $\alpha = 0.1$. Since we had enough computational resources thanks to Texas Advance Supercomputing Center, we decided to train many different models using different architectures and different hyperparameters in parallel

2.3. Batch Normalization:

Barch normalization makes the hyperparameter search problem much easier, and makes the neural network much more robust to the choice of hyperparameters and it enables us to much more easily train even very deep networks. The idea of batch norm is to apply the normalization process not only to the input layer, but also to the values even deep in some hidden layers in the neural network. We did the batch normalization before the activation function. meaning we normalized Z values rather than activations.

3. Evaluations

3.1. Benchmarking different architectures

Our evaluation runs on ResNet50 where all initially bad so we did not pursue it farther. This could have been due to its size and complexity, but because the other architectures gave us pretty good results we focused on them, specifically on InceptionV3 and VGG16.

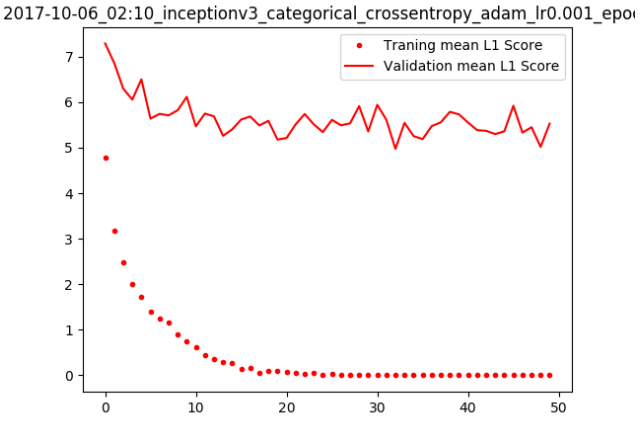


Figure 3. Inceptionv3 with Categorical Crossentropy

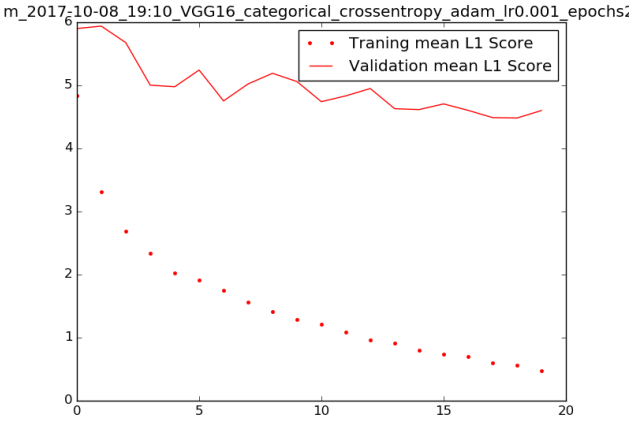


Figure 4. VGG16 with Categorical Crossentropy

3.2. Best Model

In the end, our final model was based on the VGG16 architecture using "categorical crossentropy" loss with the adam optimizer and a learning rate of 0.001.

The confusion matrix for the training and validation stages are shown. The plots are a little misleading because of the color scale, i.e. the training plot has values off the diagonal, but because of the scale ranging from 0 to 800, values near 0 appear to be zero. We were able to get the L1 error very low during training and this is reflected in the training plot. The trick was being careful not to over fit during this stage. For the validation plot, we see that the model is still centered around the diagonal, but our L1 error is 4.5 years. The confusion mostly occurs between neighboring years in the validation plot.

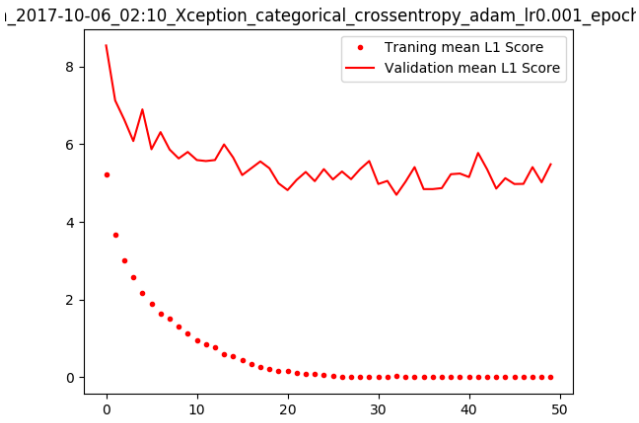


Figure 5. Xception with Categorical Crossentropy

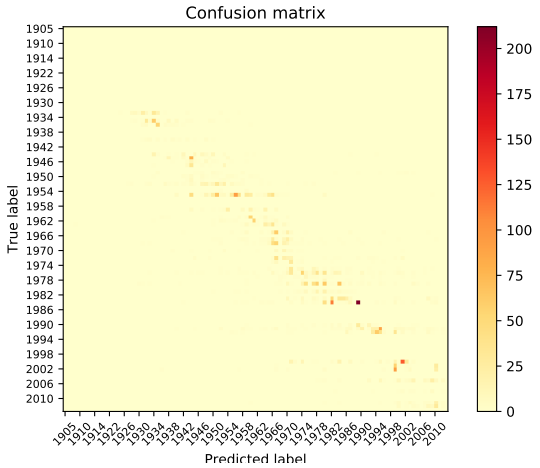


Figure 7. Confusion matrix on validation data

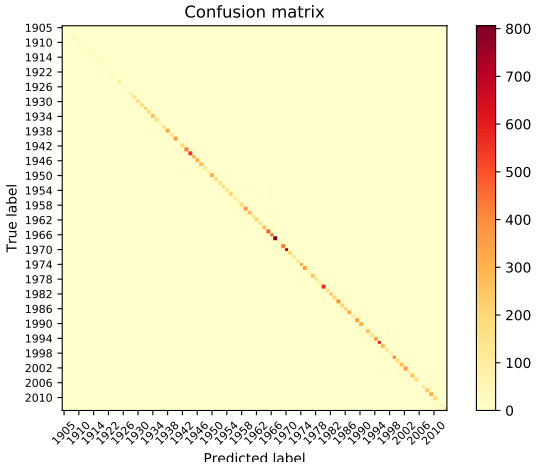


Figure 6. Confusion matrix on training data

4. Extensions

In the following section, we catalog how we visualized the important regions in the image used by the network and what layers in our network learned. Additionally we benchmark the different architectures we ran experiments on, and finally compare results for the different loss functions utilized during our tests.

4.1. Visualizations

It is often said that deep learning models are ”black boxes”, learning representations that are difficult to extract and present in a human-readable form. While this is partially true for certain types of deep learning models, it is definitely not true for convnets. The representations learned by convnets are highly amenable to visualization, in large part because they are representations of visual concepts.

To understand what our networks our learning, we utilized three very common visualization techniques:

1. visualizing intermediate convnet activations. This is great to get an idea of the meaning of individual convnet filters and how successive convnet filters transform the input

2. visualizing convnet filters. This is a great tool to understand precisely what visual patterns each filter is receptive to.

3. Visualizing heatmaps of class activations in an image. This is useful to understand which part of an image where identified as belonging to a given class.

4.1.1 Visualizing intermediate activations

This gives a view into how an input is decomposed unto the different filters learned by the network. These feature maps we want to visualize have 3 dimensions: width, height, and depth (channels). Each channel encodes relatively independent features, so the proper way to visualize these feature maps is by independently plotting the contents of every channel, as a 2D image.

For the purpose of this visualization, we will use one of the test images, say yearbook/test/F/000002.png. Here is the pre-processed image that we are going to feed into our conv net (figure 5).

For the purpose of visualizing intermediate activations, we built a multi-output model, which allows for one input and one output per layer activation. Using this network, we plot a complete visualization of all the activations in the network. Here, we show only some of these activations in our network (figure 9)

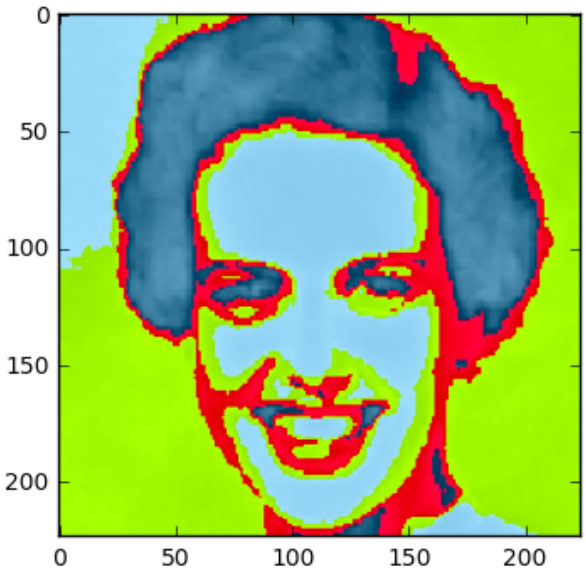


Figure 8. preprocessed test image

Figure 9. Visualizing intermediate activations

Visualizing individual intermediate activations show that, the first couple of layers act as edge detectors, and activations are retaining almost all of the information present in the initial pictures. As we go higher up, the activations become more abstract and start encoding higher level representations. These representations carry more information related to the class of the image.

4.1.2 Visualizing convnet filters

Next, we visualize the visual pattern each filter is meant to respond to. This can be done with gradient ascent in input space: applying gradient descent to the value of the input image of a convnet so as to maximize the response of a specific filter, starting from a blank input image. The resulting input image would be one that the chosen filter is maximally responsive to.

we will build a loss function that maximizes the value of a given filter in a given convolution layer, then we will use stochastic gradient descent to adjust the values of the input image so as to maximize this activation value.

Here, we will visualize the first 64 filters in VGG16 in the first layer of each convolution block block3_conv1, block4_conv1, and block5_conv1(figure 10).

This visualization shows us how each layer in a convnet learns only a collection of filters such that their inputs can be expressed as a combination of these filters. As we go higher up, the filter bank gets increasingly more complex.

4.1.3 Heatmaps of class activations

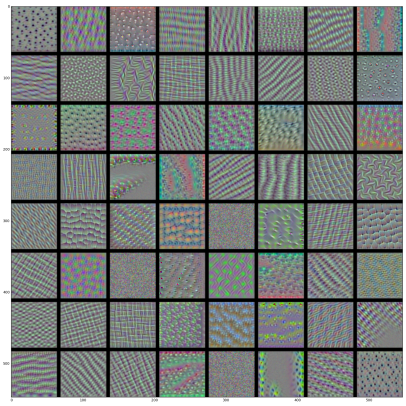
Visualizing heatmaps of class activations helps us understand which parts of a given image led a convnet to its final classification decision. A "class activation" heatmap is a 2D grid of scores associated with an specific output class, computed for every location in any input image, indicating how important each location is with respect to the class considered. This is helpful for "debugging" the decision process of a convnet, in particular in case of a classification mistake.

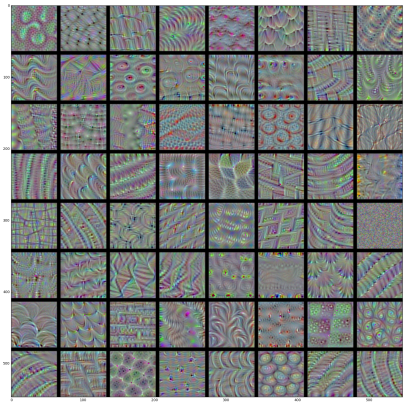
Here, we take the output feature map of a convolution layer given an input image, and weighing every channel in that feature map by the gradient of the class with respect to the channel.

4.2. Comparison of losses

To benchmark our los functions, we will show their performance on our best model, which is built on top of VGG16. Fig. 12 shows how each loss function affects the performance of this model in terms evolution of accuracy, loss and L1 score during the fine tuning phase.

The top three plots in Fig. 12 correspond to the loss function number 3 defined in section 2, i.e for every example, the loss is the sum of the cross entropy loss between the softmax output and the one hot encoded vector of true label plus the square of the difference between the true year and the predicted year. In this way we can give different weights to the loss caused by cross entropy and the loss caused by the difference between the true year and the predicted year. As can be seen, the training accuracy increases with each





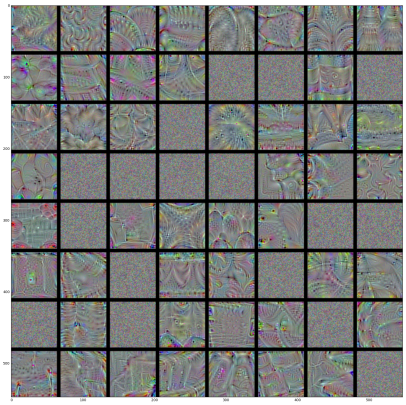


Figure 10. Visualizing convnet filters, block3_conv1 (top image), block4_conv1 (middle image), and block5_conv1 (bottom image)

Figure 11. Heatmaps of class activations

epoch and the training loss and the L1 score reduce with epoch. At the end of the 20th epoch we have fitted the training data almost perfectly and the validation L1 score has dropped to above 5 years which is satisfactory.

The middle three plots correspond to loss number 4 defined in section 2. i.e for every example, the loss is the sum of the cross entropy loss between the softmax output and the one hot encoded vector of true label plus the absolute value of the difference between the true year and the predicted year. The overall trend is the same as the previous loss except the validation accuracy does not change much during the fine tuning stage.

Finally the bottom three plots correspond to the loss number 5 defined in section 2. Essentially the loss incurred by every example is the cross entropy loss plus the L1 norm of the difference between the softmax output and the one hot encoded vector of the true label. For this loss, like the previous two the training accuracy keeps improving as we go through the epochs of training. The validation error however gets worse over the fine tuning stage. Although the val-

idation error keeps in creasing the validation mean L1 score improves and gets to a score comparable to the other two losses.

So in conclusion all the loss functions we tried were yielding good results after tuning their parameters (in this case the relative weight of the two terms of each loss) through grid search. The only loss function that did very poorly was pure MSE. This was not intuitive to us as we expected MSE to perform better. Categorical cross entropy loss does not capture the structure of the data, for example if the true label is 1950 and we make a wrong prediction, it does not make any difference with the cross validation error if we predict 2000 or 1951 but these two predictions are very different in terms of how accurate they are. This is why we were expecting to get good results by using MSE because it considers the structure of the data. Still we were able to see satisfactory results by combining the classification cross entropy loss with different sorts of regression losses, including MSE.

5. Conclusion

In this work, in order to make accurate predictions on the year photos were taken, we used several deep convolutional neural network architectures. Instead of trying to build a model from scratch and training it on the limited amount of data we had access to, we took advantage of the availability of the most successful networks trained on large amounts of images. We used transfer learning for this purpose, i.e. we took a trained ConvNet, froze the convolutional layers, replaced the fully connected layers and trained the model to learn the weights for the fully connected layers. Then we performed fine tuning on the last few convolutional layers. Using this approach we reached an accuracy of 3.0% and mean L1 error of 4.5 years on test data on our best model which is build based on VGG16.

In conclusion, using transfer learning really boosts the performance of the model if we have access to convolutional networks that are trained for the same type of data. In this case we used deep ConvNet that were trained on ImageNet so many of the features that those models have learned on other images can be useful for our task which focuses on a specific type of image, namely portraits.

The other important issue in training deep ConvNets is to come up with a suitable loss function that takes advantage of the structure in your data. In addition to the loss functions available through Keras, we devised new cost functions by combining regression and classification losses. We believe the choice of loss function has a significant effect on the performance of the model. For this problem, the best cost functions were those that used a weighted sum of cross entropy loss and some metric to measure the distance between the prediction and true label in a regression fashion.

References

[1] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[2] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[3] S. Ginosar, K. Rakelly, S. Sachs, B. Yin, and A. A. Efros. A century of portraits: A visual historical record of american high school yearbooks. *CoRR*, abs/1511.02575, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[6] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[7] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015.

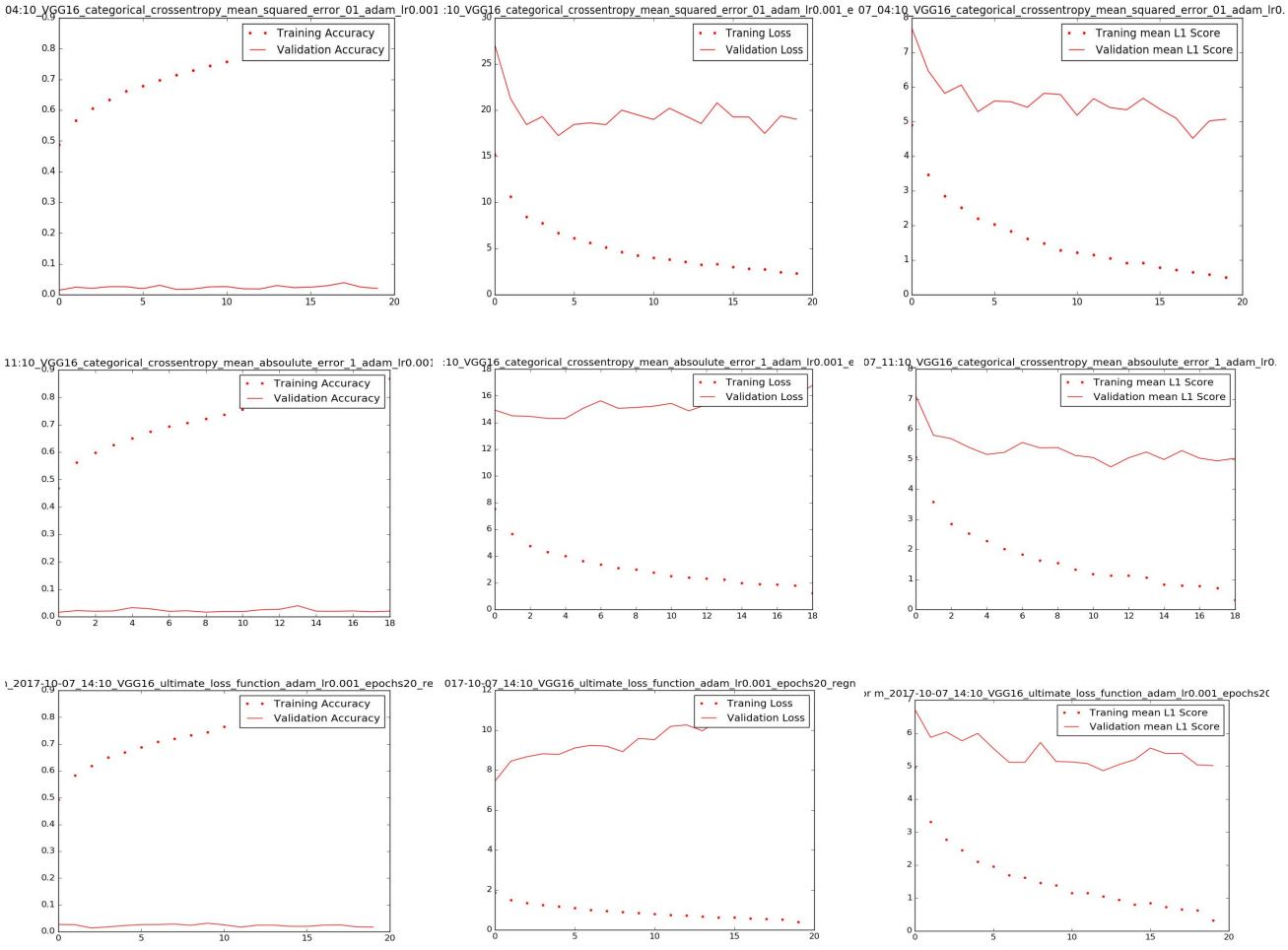


Figure 12. Evaluation of our top model on different loss functions. The top three plots correspond to categorical cross entropy plus mean squared loss, the middle three correspond to categorical cross entropy plus mean absolute error and the bottom three correspond to categorical cross entropy plus mean absolute error on years