2021

Modeling Human Visual Detection Using Deep Networks

Zach DeKraker
Michigan Technological University, zbdekrak@mtu.edu

MODELING HUMAN VISUAL DETECTION USING DEEP NETWORKS

By

Zachary B. DeKraker

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY

2021

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

College of Computing

Report Advisor:	*Dr. Timothy Havens*
Committee Member:	*Dr. Laura Brown*
Committee Member:	*Dr. Tony Pinar*
Department Chair:	*Dr. Linda Ott*

Abstract

The work in this report describes the use of machine learning to model human visual detection. This is in contrast to typical machine learning models, which seek to optimize detection performance overall, e.g., precision versus recall or F1 scores. Instead the goal is to develop models that can accurately match humans' abilities to detect objects in images. Modeling human performance in tasks has long been a benchmark for artificial intelligence (AI), from computerized chess to conversational robots. Until recently, the goal of AI has been to match human performance in complex tasks, with the assumption that human performance is an attainable goal. But there are many AI algorithms that have far surpassed humans in, for example, object detection in large image databases or games such as Go. What is different about this work is that the objective is to accurately model humans' performance in visual detection tasks, with the supporting task of knowledge discovery on how humans interpret complex images to detect objects. To accomplish this, deep learning architectures designed for image classification are adapted, extending these architectures to predict detection statistics of human observers. Results comparing human performance against machine performance indicate a strong correlation of the human and machine. Furthermore, explainable AI (XAI) methods demonstrate that the machine learning algorithms are interpreting the images in intuitive ways, which supports the notion that the algorithms are learning an accurate representation of how humans interpret

images for object detection.

Chapter 1

Introduction & Goals

Neural networks have become progressively more effective in their various domains. Many applications of machine learning and neural networks aim to exceed human capabilities in things like identifying cancerous cells [1], but in this study my objective is trying to match, rather than exceed, human performance in an observation task. That is, given a set of data that were presented to human volunteers, can the network learn which images had objects that most volunteers were able to identify, and can the network estimate how many were able to identify it? A fair amount of work has already been done on looking at various aspects of human visual perception. For instance, Yuan and Li [2] developed a testing method and a model to test various *user interface* (UI) designs and predict how long it would take users to identify a particular object amongst clutter, which would likely be of great interest to websites

that generate revenue via advertisements. Another study published by Dodge and Karam [3] compared how well humans were able to correctly classify noisy images against how well a trained network did with those same images. These are very interesting topics, but they do not look at the first of the three main focuses this project addresses, which is that of predicting the probability that a human will be able to see a given object. Development of a model that accurately mimics a large base of volunteers will greatly decrease the cost of obtaining similar probability estimates for future images (by limiting the number of necessary human volunteers), and will do so at much greater speed.

The second problem this project looks at is ordinal classification. Ordinal classification has received a fair amount of attention in literature. Multi-view max pooling [4] has been introduced as a technique to increase data augmentation capabilities and improve network performance on ordinal tasks with respect to images. For instance, improving the classification of the relative age (young, middle-aged, old) of a person in an image. Frank et al. [5] also looked at the task of ordinal classification, but took the route of training $(C - 1)$ classifiers, where C represents the number of classes, effectively making the problem one-versus-all. Perhaps the most relevant publication, by Cardoso et al. [6], describes the the most similar approach to that explored in this report, which is that of considering predicted classes farther away from the correct class as worse predictions than those closer to the correct class. Unfortunately, [6]

focuses more on calculating the error of poorly ordered class predictions. In the problem domain faced in this project, the network is not predicting a class order, so the problem becomes evaluating error where classes have some inherent numerical value such that, given an image and an observed *Probability of Detection* (P_d), e.g., 0.0, a prediction vector representing 0.1 produces a more optimal loss value than that of a prediction vector representing 0.5.

The third and final area of focus is incorporation of *explainable AI* (XAI). There are several legends about networks that achieved incredible training accuracy only to be completely thwarted by the testing set due to some minor difference the network engineers never considered. Incorporating XAI provides evidence that the network is not training on irrelevant image features.

Some of the material in this report was published in [7].

Chapter 2

Methods

2.1 Data

The data used for this project consist of renders of an environment and an object within that environment, which varied across the horizontal-axis but with a fixed vertical coordinate at half the image height. The image is then mirrored horizontally, and these two images are then presented to a human observer for a short time (~ 2 seconds). The volunteer must decide if an object exists in one of the two images, and if so, respond by pressing a button indicating which side they believe the object to be on. The probability of detection (P_d) of an object can then be calculated by dividing the total number of true positives by the sum of true positives and false negatives.

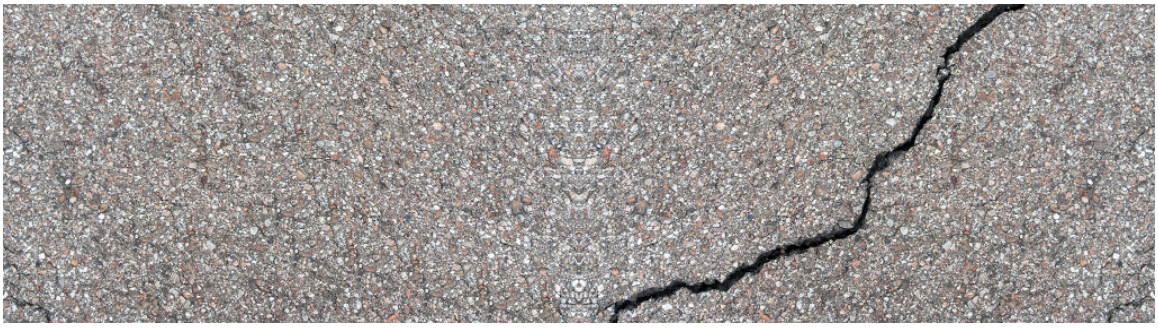

Figure 2.1: Human observation testing data - mirrored asphalt image with crack shown in right image

The aggregated P_d of the human subjects is used as the truth data for this project, while the corresponding images with an object in them are used for the prediction input. Figure 2.1 shows human testing imagery, while Figure 2.2 shows the type of image that would be used to train the network. Volunteers would be asked to identify which half of Figure 2.1 contains the crack in the asphalt, if indeed they believe there is a crack on one of the two sides. In this case, because the crack is quite obvious, it is reasonable to assume the observed P_d, corresponding to the training image in Figure 2.2, would be 1.0. Thus, Figure 2.2 and the value 1.0 would form one training sample.

2.2 Network Architecture

A *convolutional neural network* (CNN) was a natural solution to solve the problem that involved training on images. A great deal of work has already been done in

Figure 2.2: Training sample corresponding to human tested image in Figure 2.1

optimizing network architectures for image recognition problems, and so a transfer learning approach was used to compare models that leveraged Xception [8] and ResNet50 [9]. However, the high resolution of the input images, compared to the sizes of the training sets for which Xception and ResNet50 were designed, necessitated the addition of extra input layers to "down-sample" the image; albeit the down-sampling function is a learned attribute of the network. ResNet50 accepts 224×224 images as input, and Xception accepts 299×299 images, so the training image from our study was zero-padded to a square shape of 1196×1196 for Xception or 1120×1120 for ResNet50, and then down-sampled using a two-dimensional convolutional layer with 3 filters to match each network's native input size. Furthermore, the different problem domain that Xception and ResNet50 were intended for required a change in the classification layers of these networks. Both Xception and ResNet50 were designed

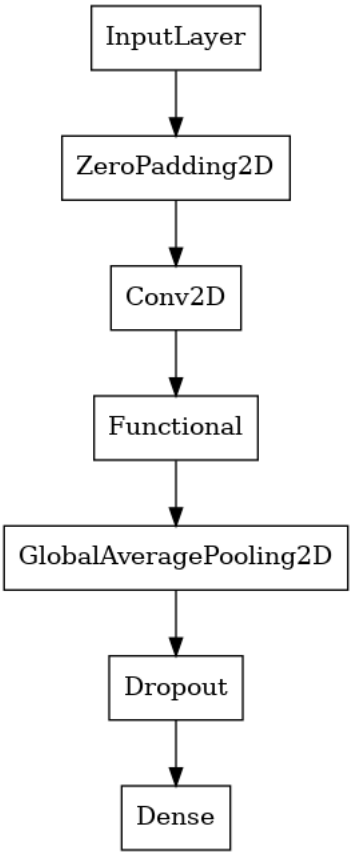

Figure 2.3: One input architecture

for multiclass classification, not regression. Thus, the final dense layers of these networks were removed and replaced with a new dense layer. For solving the problem via regression, only a single output node was used with a sigmoid activation. For the ordinal problem, 101 nodes were used with a softmax activation. These extra layers can be seen sandwiching the blocks labeled "Functional" in Figures 2.3–2.4.

Two approaches to predicting the P_d were explored. The first such paradigm involved one output neuron activated by a sigmoid function. The second paradigm was

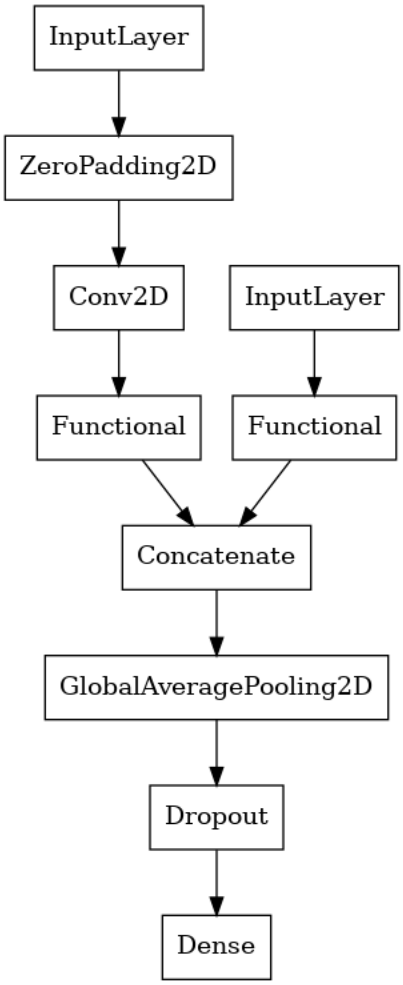

Figure 2.4: Parallel architecture

designed for ordinal classification using a softmax activation with 101 output neurons. In addition, an alternative architectural paradigm was explored, in which two instances of a pretrained network exist on parallel dataflows through the network. Thus, such a design trains on two inputs, one full resolution input, and that same full resolution input cropped to a pretrained network's native size centered around the object, based off the assumption that a perfect object detector would be able to provide such an input. This two-input architecture has been dubbed the *parallel*

path architecture. This gives a total of 8 different network architectures that are compared in Chapter 3. These architectures are referred to as X-to-Y networks, where X indicates the number of inputs, with 2 being the parallel path architecture, and Y denoting the number of outputs, with 1 referring to the sigmoid regression network and 101 referring to the ordinal classification network. Figures 2.3 and 2.4 show Keras-generated plots of the general shapes of the one- and two-input architectures, with "Functional" referring to the spot in the architecture where either a pretrained Xception or ResNet50 model would be inserted, and the final "Dense" layer designed to have either 1 or 101 output neurons.

2.3 Ordinal Classification

The idea behind ordinal classification is to get a measure of the level of confidence of the network. The way this works is by using a one-hot encoded vector instead of a single continuous value. P_d values from 0 to 1 are mapped to equally-spaced classes, say $(0, 0.1, \ldots, 0.9, 1)$. Thus, the regression problem transforms into a classification problem where the ith class represents a P_d of $\frac{i}{n}$, $i = 0, 1, \ldots, n$. A single P_d can still be predicted by taking the class with the maximum score, but the distribution of scores around the predicted class can provide a measure of confidence in the prediction, demonstrated by the amount of spread in Figures 2.5 and 2.6. In 2.6, the highest scoring classes are clustered around roughly 90%, showing that the network calculates

a very high probability that the true P_d is somewhere within $90\% \pm 5$. Conversely, 2.5 shows a distribution in which there is no particularly strong clustering of scores. To reach the same level of assurance that the network had when making the prediction for 2.6, the upper and lower bounds are much further apart, indicating a significantly lower level of certainty in the prediction. Note that the predicted probability distribution can only be interpreted as such because there is an ordering relation between all the classes. In most classification problems, there is no such meaningful relation between adjacent classes. Consider a small example using 4 classes. Given $y = [1, 0, 0, 0]$ and two theoretical predictions, $\hat{y}_1 = [0, 0, 0, 1]$ and $\hat{y}_2 = [0, 1, 0, 0]$, calculating the MSE of $\hat{y}_1$ and $\hat{y}_2$ yields $f(y, \hat{y}_1) = f(y, \hat{y}_2) = 0.5$ (where $f(\cdot)$ represents MSE), and categorical crossentropy produces $g(y, \hat{y}_1) = g(y, \hat{y}_2) \approx 16.12$ (where $g(\cdot)$ represents categorical crossentropy). The problem with existing loss functions, such as those discussed here, is that the error is calculated based on whether a class is predicted correctly or not, and pays no regard to how close the predicted class is to the true class. In traditional classification tasks, this approach makes sense. There is no ordering relation between classes that represent a boat and a pomegranate. Consequently, the network is either correct or incorrect, and there is no degree of incorrectness to consider. However, such an ordering relation does exist in the ordinal classification task discussed here. To resolve this issue, a new loss function called Squared Ordinal

Loss is introduced, which is defined as

$$\mathcal{L}_{ORD} = \sum_{k=1}^{K}((Cy)_k \hat{y}_k)^2, \tag{2.1}$$

where y is a one-hot encoded vector (rounded to the nearest percentage point) indicating what the observed P_d is in terms of classes, and $\hat{y}$ is the network's prediction. C is a $K \times K$ weight matrix that allows for the ordering relation between classes to be accounted for by linearly applying more weight to classes farther away from the true class according to $C_{ij} = |i - y_j|$, though one could imagine using any weight mapping appropriate for their problem. Multiplying C by y retrieves the correct weight vector from C and applies those weights to the prediction. Summing the squares of the weighted predictions provides a loss for the training sample. Considering the above example, where $y = [1, 0, 0, 0]$, the weight matrix is

$$C = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 1 & 0 \end{bmatrix},$$

and multiplying by y reduces the matrix into the appropriate weight vector $C_0 = [0, 1, 2, 3]$. Thus, a prediction of y_1 should be calculated to be much worse than a prediction of y_2. Indeed, calculating these losses shows that $\mathcal{L}_{ORD}(y_1) = 0+0+0+3 = 3 > \mathcal{L}_{ORD}(y_2) = 0+1+0+0 = 1$. A sample weight vector extracted from the weight

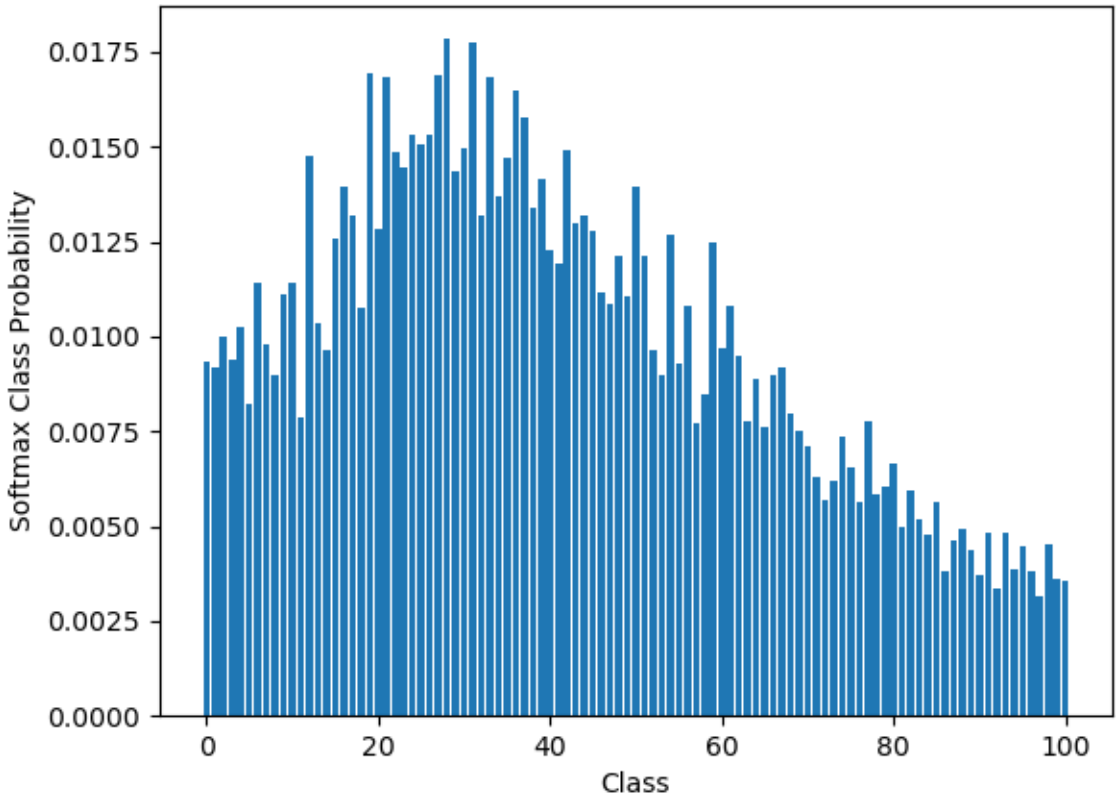

Figure 2.5: Low confidence ordinal prediction

matrix used during training is shown in Figure 2.7 with a P_d truth of $\hat{y} = 0.40$. At a prediction of $y = 0.40$ (the 40th class), the weight is 0, and it increases linearly as classes continue to grow more incorrect to a maximum weight of 60 for the 101st class, which corresponds to a P_d of 100%. Now that the ordinal loss function is defined, the application of an XAI approach to provide interpretability of the trained deep network will be discussed next.

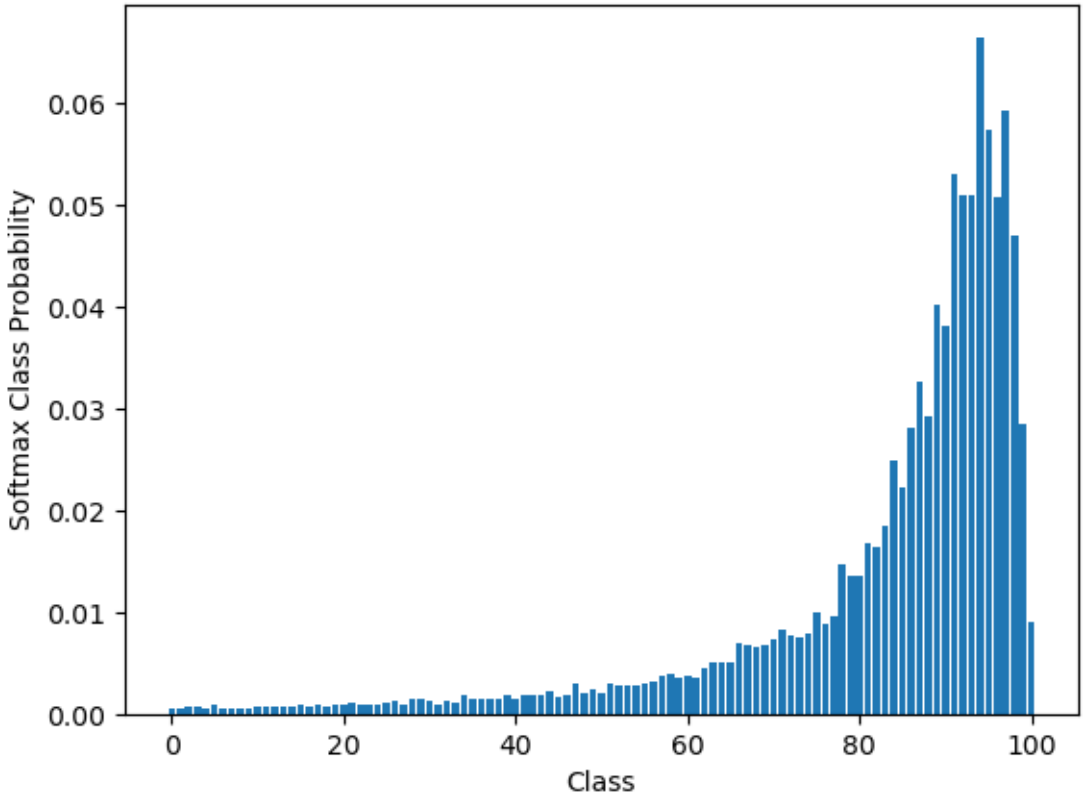

Figure 2.6: High confidence ordinal prediction

2.4 Class Activation Mapping

Class activation mapping (CAM) [10] is an XAI technique that can be used to visualize the pixels that most impact a network's prediction. Shown in Figure 2.8 is an example heatmap generated using CAM on an unmodified Xception network. As this figure shows, the pixels in the image that contributed most significantly to the classification—shown as white and gray colors—are concentrated around the rivet and blades.

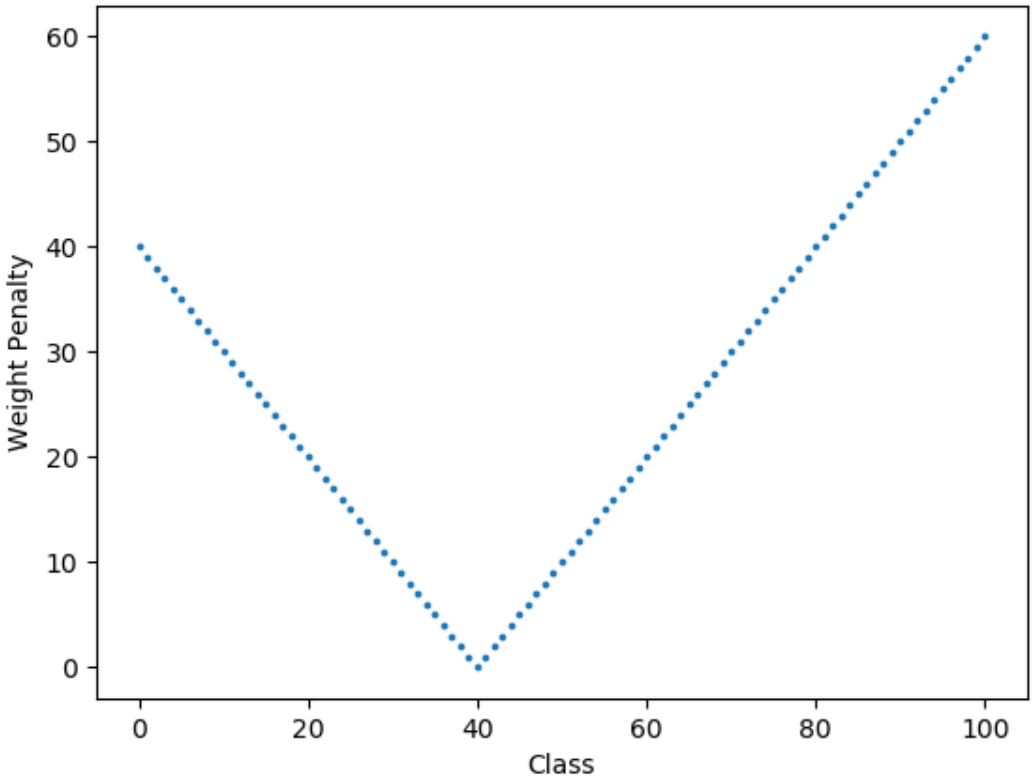

Figure 2.7: Sample Weight Vector for a 0.40 P_d

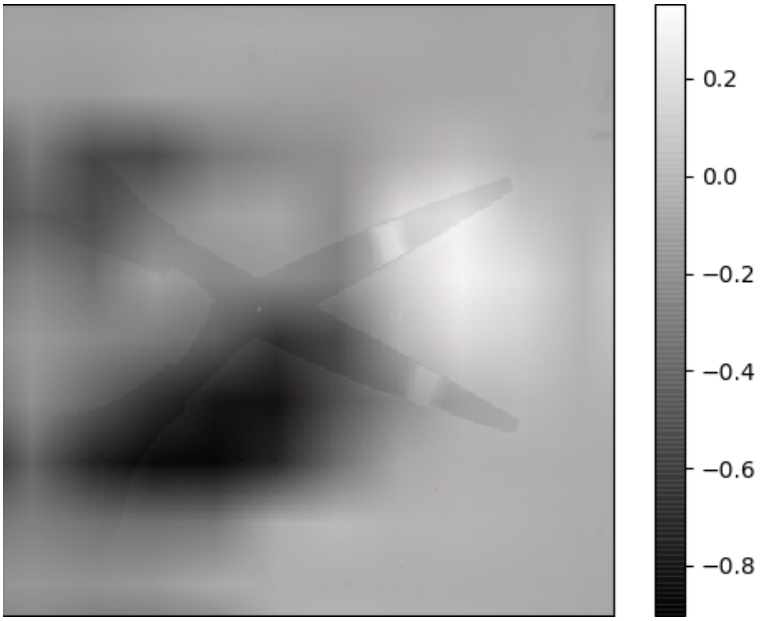

Figure 2.8: CAM with Xception on a pair of scissors

This technique was applied to the data using a 1-to-1 Xception-based architecture, and in the majority of cases, CAM produced a circular heatmap centered on the object which Figure 2.11 demonstrates. The instances in which it did not were images where the object was nearly invisible, coinciding with a P_d near 0.0, which Figure 2.12 demonstrates. The difference between these two high- and low-P_d heatmaps is rather stark. The network is very tightly focused on one area of the image that has a P_d of 1.0, and is not particularly focused at all in the image with a low corresponding P_d. Finally, to visualize the pixels that tended to be weighted most highly by the network, an input image where all pixels had a value of 1 was used. This CAM heatmap is depicted in Figure 2.9, and shows that the network weights pixels towards the side of the input more heavily. A higher-resolution version shown in 2.10 upsampled to 1196×1196 via bi-linear interpolation shows a smoother distribution of the weights. The distributions shown agree with the majority of the object positions, which tended to be away from the center of the image; see Figure 2.13 for a histogram of the object locations in the image data set.

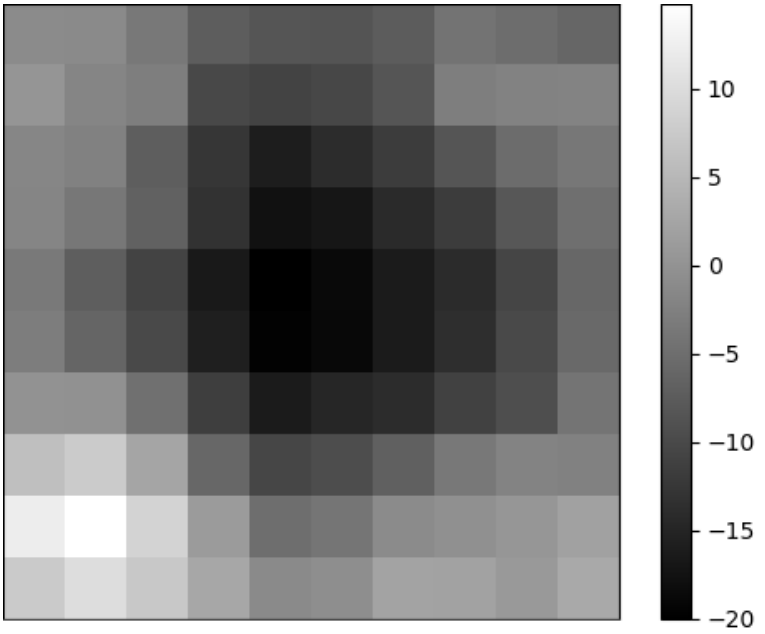

Figure 2.9: CAM heatmap produced from a uniform image

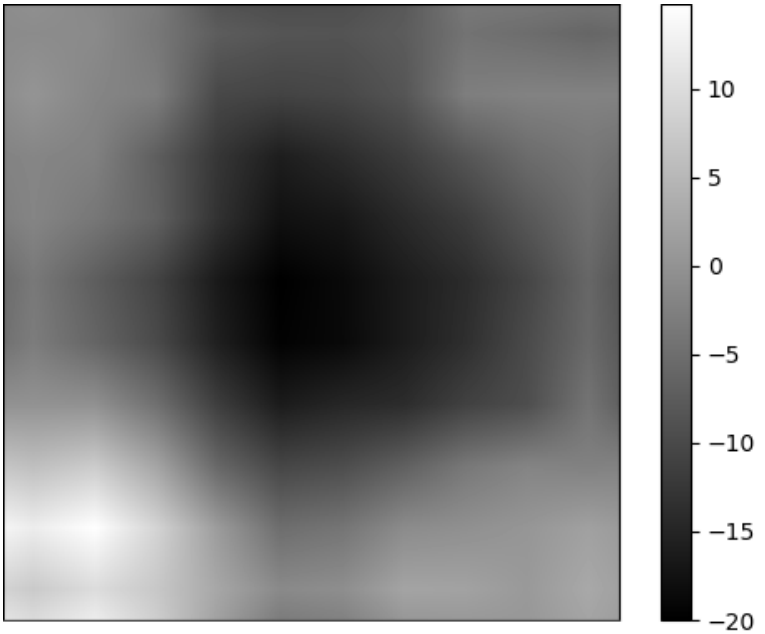

Figure 2.10: Upsampled version of Figure 2.9

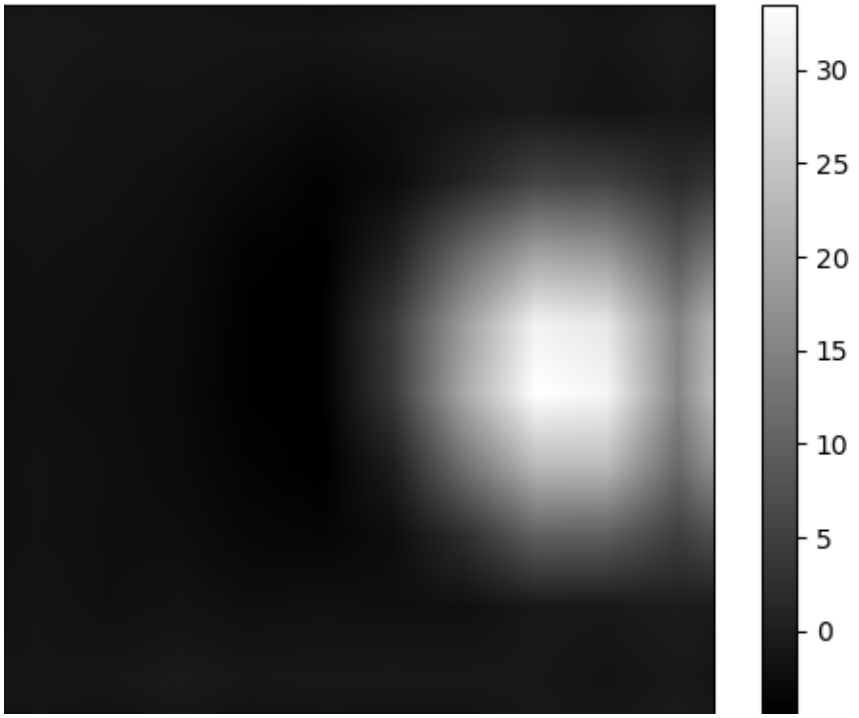

Figure 2.11: High P_d (1.0) CAM Heatmap

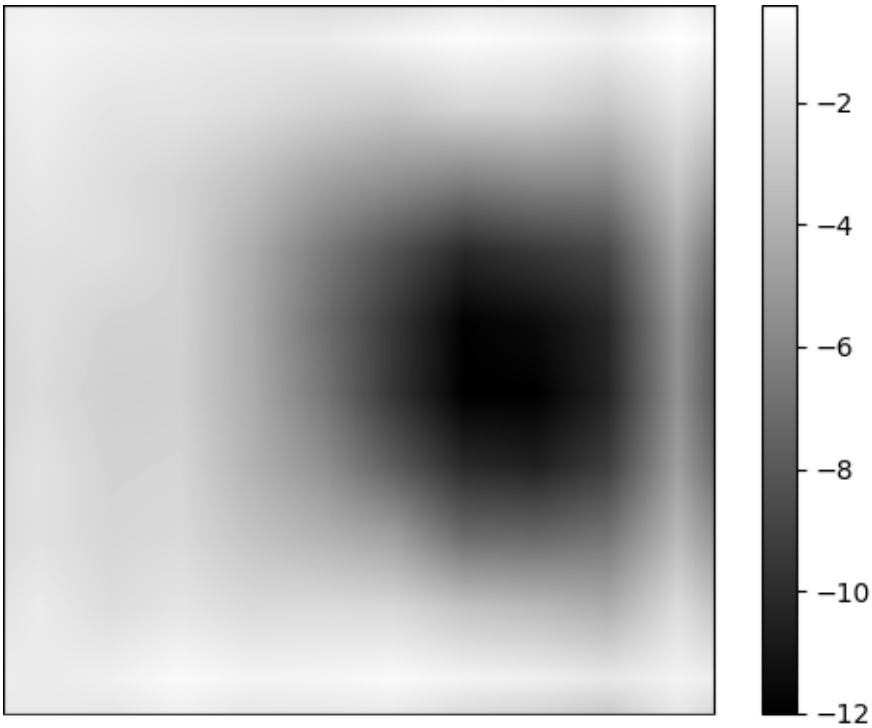

Figure 2.12: Low P_d (0.0) CAM Heatmap

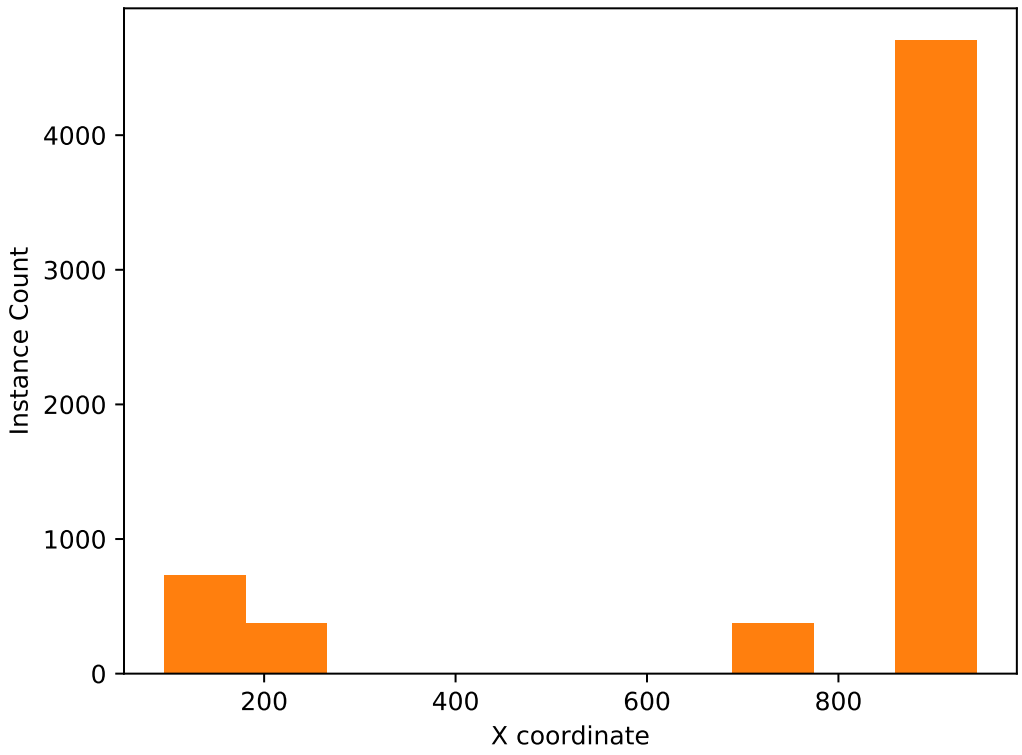

Figure 2.13: Histogram of horizontal X coordinate of object locations in image data set

Chapter 3

Results and Discussion

As mentioned in Section 2.2 there are 8 different network architectures being compared here, each with different base networks, different numbers of inputs, and different methods of classification being used. Each combination of these variables produce one architecture, which was trained and tested using 5-fold cross validation for 80 epochs, and with a decaying learning rate that started at 0.01 and decayed by an order of magnitude every 20 epochs to a minimum of 1×10^{-5} at the start of epoch 60. These parameters, including the order and contents of each fold, were held constant across each trial to help ensure consistency in the results. In addition to comparing the architectures instantiated with the aforementioned variables, various weight initialization strategies are also analyzed in an effort to determine how such strategies affect the performance of the parallel architectures. An interesting note is

that peak performance may include a prediction that indicates maximum uncertainty, i.e. a $P_d \approx 0.5$. The truth data in this study are calculated from the mean of human performance, which will never approach the same surety as truth data that, for instance, classifies an image as a cat or a dog. Because the probability of detection can vary depending on where the observer happens to be looking, eye or mental fatigue and the existence of outliers, among other factors, a perfect prediction will always be rare, so some measure of distance between prediction and truth must be used in every metric.

Tables 3.1 and 3.2 summarize the *mean-squared-error* (MSE) calculated per fold of the 8 different architectures tested. The MSE of the regression-style networks (the networks that used MSE to converge to a single P_d prediction in the continuous interval $[0, 1]$) was calculated using

$$\mathcal{L}_{MSE} = \frac{1}{n}\sum_{i=0}^{n}(y - \hat{y})^2 \tag{3.1}$$

where y represents the truth and $\hat{y}$ the prediction. The prediction and truth vectors of the ordinal classification-style architecture (the networks which use a classification-style approach with 101 classes representing each percentage point between 0 and 100 inclusive) were condensed into single P_d by taking the index of the class with the greatest probability, yielding the truth index (i_t) and the predicted index (i_p). MSE

Table 3.1
Xception Architectures: MSE on Validation Set for 5 Folds

	1-to-1	1-to-101	2-to-1	2-to-101
Fold 1	0.154	0.033	0.025	0.033
Fold 2	0.168	0.015	0.047	0.023
Fold 3	0.158	0.020	0.044	0.021
Fold 4	0.103	0.020	0.035	0.021
Fold 5	0.136	0.033	0.039	0.020
Mean MSE	0.144	0.024	0.038	0.024
Standard Deviation	0.023	0.007	0.008	0.005

Table 3.2
ResNet50 Architectures: MSE on Validation Set for 5 Folds

	1-to-1	1-to-101	2-to-1	2-to-101
Fold 1	0.058	0.184	0.063	0.207
Fold 2	0.095	0.190	0.052	0.245
Fold 3	0.143	0.178	0.041	0.304
Fold 4	0.098	0.245	0.054	0.189
Fold 5	0.153	0.177	0.051	0.188
Mean MSE	0.109	0.195	0.052	0.226
Standard Deviation	0.035	0.026	0.007	0.044

can then be applied via equation 3.1 where $y = \frac{i_t}{100}$ and $\hat{y} = \frac{i_p}{100}$.

The experiment consisted of setting up a trial for each network architecture and different methods of instantiating the weights of those that are parallel. Each trial consisted of training a network 5 times, with each training instance holding back a different fold to use as a validation set. These folds were determined by splitting up a randomized collection of references to the data into five unique and disjoint sets. These sets were then serialized so that the same data in the same order could be used across the different architectures. Each architecture was trained for 80 epochs on each of the five folds, and the mean and standard deviation of the MSEs from the

Table 3.3
Means Over Fold MSE's for Parallel Architecture Weight Initialization Strategies

	Ordinal		Regression	
	ResNet50	Xception	ResNet50	Xception
Random	0.2898	0.2235	0.2005	0.1138
Imagenet	0.1804	0.0204	0.0440	0.1185
Pretrained	0.2258	0.0165	0.0324	0.1120

five trials were calculated as overall measures of performance. One very important distinction between the parallel and non-parallel architectures is that some of the parallel architectures used base networks initialized with weights from the trained non-parallel network, though this is not the case for all trials.

Three methods of initialization were explored for the parallel architectures. The first, *random*, simply randomly initialized all weights. The second, *imagenet*, loaded each subnetwork of the parallel architecture with the pretrained imagenet weights for that network. Finally, *pretrained* refers to a weight initialization that used the weights of the matching, fully trained non-parallel network. The comparison among these results is made in Table 3.3. For brevity, only the arithmetic mean of the validation loss across the five folds is shown.

To compare the results across the different architectures, an unpaired two-sample T-test is used between the sample of means computed over the folds for the architectures being compared. The T-test shows the statistical significance of the possible different performance, highlighting which network (and weight initialization variant

Table 3.4
Shaprio-Wilk P-values of Weight Initialization Strategies

	Xception		ResNet50	
	Regression	Ordinal	Regression	Ordinal
Random	0.858	0.000	0.463	0.491
Imagenet	0.728	0.245	0.609	0.125
Pretrained	0.690	0.467	0.821	0.003

Table 3.5
Shapiro-Wilk P-values of all Explored Architectures

Architectures	Xception	ResNet50
1-to-1	0.419	0.595
2-to-1	0.810	0.010
1-to-101	0.2128	0.786
2-to-101	0.041	0.196

for the parallel instances) performs most optimally. Tables 3.4 and 3.5 show the Shapiro-Wilk P-values calculated for each weight initialization strategy and architecture, respectively, to show that each set of errors is roughly sufficiently normal to proceed with the T-tests. There are some exceptions, but the data sets are generally sufficiently normal for this test to proceed.

The tables showing the T-test comparisons display the statistical difference between the mean of the first item listed and the mean of the second (in other words, the statistical difference $\mu_1 - \mu_2$) at 95% confidence. The value contained in the cell is a *confidence interval* (CI), showing what the upper and lower bounds are of that statistical difference at 95% confidence. If $\mu_1 > \mu_2$ (where $>$ represents statistically larger), the lower bound of the CI will be greater than 0. If $mu_1 < \mu_2$ (where $<$ represents statistically lesser) the upper bound of the CI will be less than 0. If

$\mu_1 == \mu_2$ (where == represents statistically equivalent), 0 will fall between the CI's upper and lower bounds. The cells which show either that a parallel design outperformed the single-input design or that ordinal loss outperformed MSE are green. Those that show the opposite are highlighted in red. In the instance that there is no difference, the text will be black.

The first comparisons will be between the various weight initialization strategies explored. The results shown in Tables 3.6 and 3.7 demonstrate that for most pairings there was no significant difference in mean performance between any pair of initialization methods. This somewhat surprising as the *pretrained* initialization strategies effectively received more training time, via each half of the parallel network getting instantiated from weights obtained from a non-parallel network already trained for 80 epochs. In general, it appears that the weight initialization strategy does not greatly affect the loss. The only situation in which an improvement is seen is in the parallel ResNet50 regression architecture, where randomized initialization was outperformed by alternative strategies. Given these results, it appears that weight initialization does not play an important role, and the remaining architectural comparisons will take place between networks instantiated with *imagenet*.

Tables 3.8 and 3.9 show the comparisons between the regression and ordinal networks, and between the parallel and non-parallel architectures respectively. Table 3.8 reveals that for all cases except one, ordinal prediction and training results in

Table 3.6
95% CI's of T-tests comparing Weight Initialization Strategies for Xception

Initializations Compared	CI Bounds	
	Regression	Ordinal
Pretrained vs Imagenet	(-0.015, 0.002)	(-0.013, 0.006)
Pretrained vs Random	(-0.017, 0.013)	(-0.325, -0.089)
Imagenet vs Random	(-0.010, 0.019)	(-0.321, -0.085)

Table 3.7
95% CI's of T-tests comparing Weight Initialization Strategies for ResNet50

Initializaitons Compared	CI Bounds	
	Regression	Ordinal
Pretrained vs Imagenet	(-0.064, 0.155)	(-0.425, 0.262)
Pretrained vs Random	(-0.182, -0.153)	(-0.217, 0.089)
Imagenet vs Random	(-0.169, -0.143)	(-0.326, 0.361)

Table 3.8
95% CI's of T-tests comparing the Ordinal Architectures to the Regression Architectures

Architectures Compared	Xception	ResNet50
1-to-1 vs 1-to-101	(0.088, 0.151)	(-0.136, 0.035)
2-to-1 vs 2-to-101	(0.003, 0.025)	(-0.235, -0.114)

no worse behavior than a regression-style network; the one exception being parallel ordinal ResNet50. However, Table 3.9 shows that using a parallel network never results in worse behavior. In every case it is either a performance improvement or it makes no difference. From this, parallel ordinal Xception appears to be the best network architecture of all those explored in this report, and indeed, looking at the resulting means and standard deviations presented in Tables 3.1 and 3.2 reinforces this suggestion. By performance metrics alone, ordinal loss is statistically at least as good as MSE, and due to the extra information it can convey about the uncertainty of a prediction, one could claim it is a more useful loss function than MSE.

Table 3.9
95% CI's of T-tests comparing Parallel to Non-Parallel

Architectures Compared	Xception	ResNet50
1-to-1 vs 2-to-1	(0.075, 0.138)	(0.009, 0.105)
1-to-101 vs 2-to-101	(0.024, 0.011)	(-0.093, 0.029)

Chapter 4

Conclusion and Future Work

In this report, existing networks optimized for image classification tasks are leveraged in a transfer learning approach in order to predict human visual detection performance. Utilizing a weight visualization technique called CAM rationalized network performance and indicated the networks behave as expected and as desired. Experimenting with various weight initialization strategies showed that how weights are initialized is irrelevant, and testing the remaining permutations of variables against each other to find the most optimal network revealed that parallel instances of the Xception architecture utilizing the ordinal loss function outperformed the alternatives. Finally, and most significantly, a new loss function is explored that attempts to classify an image's probability of detection as one of 101 discrete ordered values representing discrete bins of percentage points, allowing for a single prediction to

simultaneously provide information about the appropriate class and a measure of confidence in that prediction.

In future work, exploring how scaling the number of parallel networks impacts performance may yield favorable results, given that the parallel architectures generally outperformed the non-parallel architectures. Further investigation into how the distribution of classified P_d "bins" can inform epistemic uncertainty, and the feasibility of transforming that hyperparameter into a variable that can be learned may also lead to interesting finds.

References

[1] Z. Xu and J. Huang, "Efficient lung cancer cell detection with deep convolution neural network," in *International Workshop on Patch-based Techniques in Medical Imaging*, 2015.

[2] A. Yuan and Y. Li, "Modeling human visual search performance on realistic webpages using analytical and deep learning methods," in *CHI Conference on Human Factors in Computing*, 2020.

[3] S. Dodge and L. Karam, "A study and comparison of human and deep learning recognition performance under visual distortion," in *26th ICCCN*, 2017.

[4] C. Zhang, X. Xu, and C. Zhu, "Image ordinal classification with deep multi-view learning," *Electronics Letters*, vol. 54, pp. 1280–82, 2018.

[5] E. Frank and M. Hall, "A simple approach to ordinal classification," in *European Conference on Machine Learning*, pp. 145–56, 2001.

[6] J. Cardoso and R. Sousa, "Measuring the performance of ordinal classification," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 25, pp. 1173–95, 2011.

[7] S. Whitaker, Z. DeKraker, A. Barnard, T. Havens, and G. Anderson, "Uncertain inference using ordinal classification in deep networks for acoustic localization," in *IJCNN*, 2021.

[8] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *IEEE CVPR*, 2017.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE CVPR*, 2016.

[10] B. Zhou, A. Khosla, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *IEEE CVPR*, 2016.