Modularbeit Deep Learing - Image Denoising

Bartolovic Eduard
Hochschule München
München, Deutschland
eduard.bartolovic0@hm.edu

***Zusammenfassung*—Im Rahmen dieser Modularbeit wurde das Image Denoising mit Deep Learning behandelt. Ziel ist es, ein Modell zu entwickeln, welches verrauschte Bilder aufnimmt und diese bestmöglich rekonstruiert. Hierfür wurden verschiedene Netzarchitekturen und Verfahren getestet. Es zeigte sich, dass ein Convolutional Autoencoder mit Skip-Connections sehr gut für diesen Einsatzzweck geeignet ist. Diesem Modell war es möglich, ein sehr stark verrauschtes Bild aus einem eigens angefertigten Datensatz mit sehr geringem Fehler wieder zu rekonstruieren.**

I. Einleitung

Die Aufgabe, die im Rahmen dieser Modularbeit bearbeitet wird, ist das Image Denoising. Ziel ist es, ein Bild mit Rauschen bestmöglich wieder zu rekonstruieren, ohne großen Qualitätsverlust zu erleiden. Für diese Aufgabe wurde Deep Learning genutzt. In den folgenden Kapiteln wird beschrieben, wie dieses neuronale Netz trainiert worden ist und wie gut es seine Aufgabe erfüllt. Verwendet wurde hierfür Keras und Tensorflow 2.7.

II. Datensatz

Als Datensatz wurde ein selbst angefertigter Datensatz verwendet, welcher im Rahmen des Projekts ”Autonomes Fahren” entstand. In der Abbildung 1 sind zwei Beispiele aus dem Datensatz abgebildet. Hierbei befindet sich eine Kamera auf dem Fahrzeug und filmt die Szene. Es gibt eine Vielzahl von verschiedenen Szenen. Alle Bilder haben die Gemeinsamkeit, entweder die blaue Motorhaube oder die schwarze Karosserie im unten Bildbereich zu haben. Dieses Merkmal sollte hier sehr einfach zu entrauschen sein. Der Datensatz enthält 5000 Farbbilder, die jeweils eine Auflösung von 1280x960 besitzen. Aufgrund von limitierten Rechenressourcen von 32 GB RAM wurden die Bilder auf eine Auflösung von 300x300 runterskaliert.

III. Pre-Processing

Damit das Resultat wirklich aussagekräftig ist, ist der Datensatz nach der besten Praxis in einen Trainingsdatensatz mit 4000 Bildern (80%), Validierungsdatensatz mit 500 Bildern (10%) und Testdatensatz mit 500 Bildern (10%) aufgeteilt worden. Eine Crossvalidation ist aufgrund des nötigen Rechenaufwands nicht möglich.
Anschließend wird das Rauschen mit einer Funktion $C(\tilde{x}|x)$ auf die Bilder angewendet. Hierfür wurde *Gaussian Noise* genutzt, welches Rauschen mit der folgenden Wahrscheinlichkeitsverteilung erzeugt.

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

Um die maximalen Möglichkeiten der verwendeten Technik zu sondieren, sind Bilder sehr stark verrauscht. In der Abbildung 1 befinden sich zwei Bildpaare, die jeweils das Original und das verrauschte Bild zeigen. Im Anschluss werden alle Farbkanäle der Bilder von dem Wertebereich 0 bis 255 in einen Wertebereich zwischen 0 und 1 skaliert, da dies die Performance des Netzes steigern kann. Eine weitere Verarbeitung wie eine Korrektur der Daten ist nicht notwendig. Eine Datenaugmentierung ist aufgrund der reichlich vorhandenen Daten nicht nötig.

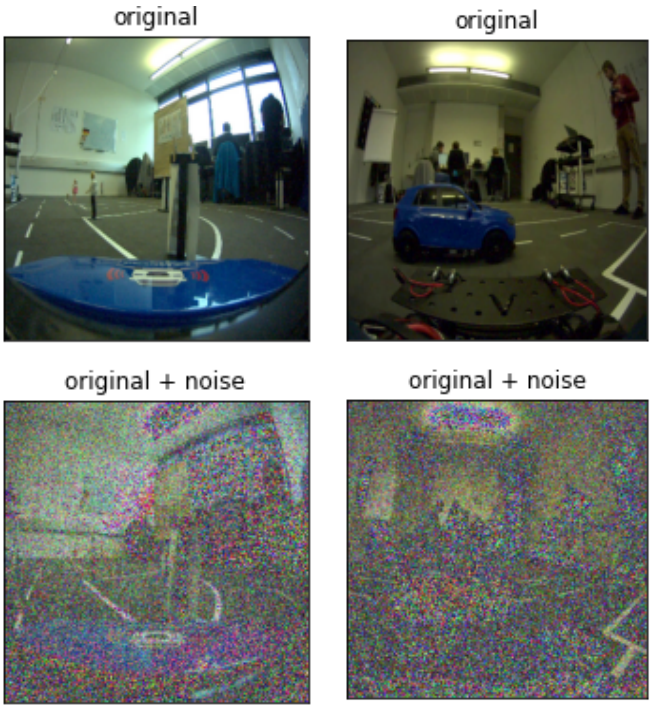

Abbildung 1: Zwei Bildpaare aus dem Datensatz, die jeweils das Original und das verrauschte Bild zeigen

IV. Autoencoder Basics

Als neuronales Netzwerk wurde ein Autoencoder verwendet. Dieser eignet sich ideal für diesen Einsatzzweck und wird häufig für genau diesen Zweck in der Literatur eingesetzt [1]. Die markante Eigenschaft eines Autoencoders ist der Flaschenhals in der Mitte. Durch diesen muss das Netz die grundlegende Struktur von den Eingangsbildern x verstehen und damit in der Lage sein, eine Rekonstruktion r anzufertigen.

Ohne diesen Flaschenhals würde das Netz versuchen, nur die Daten zu kopieren und nicht lernen, markante Merkmale nicht erkennen. Jeder Autoencoder besteht aus zwei Netzkomponenten. Einmal einem Encoder $h = f(x)$ und einen Decoder $r = g(h)$. Der bereits angesprochene Flaschenhals befindet sich zwischen diesen beiden Komponenten. Ein regulärer Autoencoder minimiert die Lossfunktion:

$$L(x, g(f(x)))$$

Dies ist aber für einen Denoising Autoencoders ungeschickt, da dieser nicht das verrauschte Ursprungsbild $\tilde{x}$ rekonstruieren soll, sondern das Rauschen aus $\tilde{x}$ entfernen soll. Daher muss die Lossfunktion

$$L(x, g(f(\tilde{x})))$$

miniert werden [1]. Die Abbildung 2 verbildlicht diesen Ablauf noch einmal.

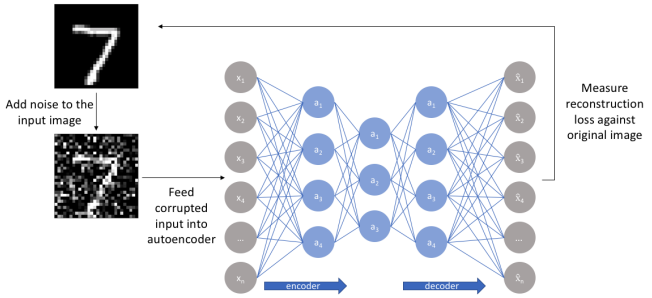

Abbildung 2: Vereinfachte Funktionsweise eines Image Denoising Autoencoders [5].

V. Netzwerkarchitektur

Es gibt nahezu unendlich viele Möglichkeiten einen Autoencoder zu gestalten. Es kann die Anzahl, Größe und Art der Schichten verändert werden. Ein wichtiger Aspekt ist die Größe des Flaschenhalses. Aktivierungsfunktionen und Regularisierungsmethoden sind weitere Stellschrauben, um Modelle zu konfigurieren. Für die Aufgabe Image Denoising ist es sinnvoll, Convolutional Autoencoder zu verwenden, da diese besonders gut in der Lage sind, Beziehungen zwischen den Pixeln eines Bildes zu erfassen [1].
Als Erstes wurde eine einfache Struktur aus einem Keras Blog Post gewählt [2]. Das Netz selbst besteht aus mehreren Convolutional Layer, die in Richtung der Mitte (Flaschenhals) in ihrer räumlichen Auflösung kleiner werden. Zwischen den Convolutional Layer im Encoder befinden sich MaxPooling Layer und im Decoder UpSampling Layer. Encoder und Decoder sind symmetrisch aufgebaut. Als Aktivierungsfunktion wird *Rectified Linear Unit (ReLU)* verwendet. In der letzten Schicht wird die Sigmoid Aktivierungsfunktion verwendet.
In weiteren Versuchen wurde die Größe des Flaschenhalses verkleinert und vergrößert. Die ursprüngliche Verengung auf 25% im Vergleich zu der Eingangsschicht lieferte die besten Ergebnisse. Alternative Aktivierungsfunktionen wie Leaky ReLU die das Problem von toten Neuronen verringern können, konnten keine signifikante bessere Performance bieten.
Regularisierungtechniken wie Batch Normalization, Dropout und Kernel Regularizer wurden gegen Overfitting eingesetzt. Nur Batch Normalization zeigte positive Effekte.
Da Autoencoder wegen ihrer Faltungsschichten Probleme haben können, aus dem Flaschenhals heraus die Bilder zu rekonstruieren, können Skip-Connection Informationen liefern die den Flaschenhals umgehen und helfen, das Bild zu rekonstruieren [3]. Zwei solche Skip-Connections wurden wie in diesem Blog Eintrag [4] eingebaut. Diese sorgten für deutlich bessere Ergebnisse.
Als Letztes wurde einfachen Upsampling Layer mit Transposed Convolution Layer ersetzt, welche ebenfalls beim rekonstruieren aus dem Flaschenhals helfen können. Dies Verbesserte das Modell ebenfalls.
Der finale Modellplan des Netzes ist in der Abbildung 3 abgebildet.

Layer (type)	Output Shape	Param #	Connected to
input_15 (InputLayer)	[(None, 300, 300, 3)	0	
conv2d_116 (Conv2D)	(None, 300, 300, 64)	1792	input_15[0][0]
batch_normalization_36 (BatchNo	(None, 300, 300, 64)	256	conv2d_116[0][0]
conv2d_117 (Conv2D)	(None, 300, 300, 64)	36928	batch_normalization_36[0][0]
max_pooling2d_30 (MaxPooling2D)	(None, 150, 150, 64)	0	conv2d_117[0][0]
conv2d_118 (Conv2D)	(None, 150, 150, 64)	36928	max_pooling2d_30[0][0]
batch_normalization_37 (BatchNo	(None, 150, 150, 64)	256	conv2d_118[0][0]
conv2d_119 (Conv2D)	(None, 150, 150, 128	73856	batch_normalization_37[0][0]
max_pooling2d_31 (MaxPooling2D)	(None, 75, 75, 128)	0	conv2d_119[0][0]
conv2d_120 (Conv2D)	(None, 75, 75, 256)	295168	max_pooling2d_31[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 150, 150, 128	295040	conv2d_120[0][0]
conv2d_121 (Conv2D)	(None, 150, 150, 128	147584	conv2d_transpose_4[0][0]
batch_normalization_38 (BatchNo	(None, 150, 150, 128	512	conv2d_121[0][0]
conv2d_122 (Conv2D)	(None, 150, 150, 64)	73792	batch_normalization_38[0][0]
add_12 (Add)	(None, 150, 150, 64)	0	batch_normalization_37[0][0] conv2d_122[0][0]
conv2d_transpose_5 (Conv2DTrans	(None, 300, 300, 64)	36928	add_12[0][0]
conv2d_123 (Conv2D)	(None, 300, 300, 64)	36928	conv2d_transpose_5[0][0]
batch_normalization_39 (BatchNo	(None, 300, 300, 64)	256	conv2d_123[0][0]
conv2d_124 (Conv2D)	(None, 300, 300, 64)	36928	batch_normalization_39[0][0]
add_13 (Add)	(None, 300, 300, 64)	0	conv2d_117[0][0] conv2d_124[0][0]
conv2d_125 (Conv2D)	(None, 300, 300, 3)	1731	add_13[0][0]

Total params: 1,074,883
Trainable params: 1,074,243
Non-trainable params: 640

Abbildung 3: Modellbauplan der finalen Architektur

VI. Hyperparameter Optimization

Eine Hyperparameteroptimierung kann die Leistung eines Netzes erheblich verbessern. Einflussreiche Parameter sind die Lossfunktion, der Optimierer, die Lernrate und die *Mini Batch Size*.

A. Lossfunktion

Zu Beginn muss die Frage geklärt sein, wie überhaupt zwei Bilder verglichen werden können. Deshalb werden in dieser Modularbeit mehrere Metriken untersucht, die sich hierfür eignen könnten.
Die einfachste Metrik zur Berechnung des Fehlers zwischen zwei Farbbildern ist der *Mean Squared Error (MSE)*.

$$MSE(r,p) = \frac{1}{W * H} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} \sum_{c=0}^{C-1} [Y_r(x,y,c) - Y_p(x,y,c)]^2$$

Das W ist die Breite und H ist die Höhe des Bildes. Y_r und Y_p entsprechen den Pixelwerten an den entsprechenden Stellen. Diese Funktion ist eine einfach und schnell zu berechnende Metrik, die die einzelnen Pixel der Bilder vergleicht. Ist der *MSE* null, dann sind beide Bilder identisch. Ein Problem an dem *MSE* Loss ist, dass zwei Bilder, die einen hohen Score erreichen für die menschliche Wahrnehmung noch sehr unterschiedlich aussehen können. Deshalb macht es Sinn, alternative Metriken in Betracht zu ziehen, die sich mehr an der menschlichen Wahrnehmung orientieren. Eine dieser Metriken ist das *Peak signal-to-noise ratio (PSNR)*.

$$PSNR(r,p) = 10 \cdot \log_{10} \frac{MAX_I}{MSE_{r,p}}$$

MAX_I ist der maximale Intensitätswert den ein Pixel einnehmen kann. Ein höherer Wert ist besser. Je ähnlicher sich die beiden Bilder sind, desto mehr nähert sich der Score gegen unendlich an, da der *MSE* Fehler im Nenner null wird [6]. Eine weitere Metrik ist das Strukturelle Ähnlichkeitsmaß (SSIM).

$$\text{SSIM}(r,p) = \frac{(2\mu_r\mu_p + c_1)(2\sigma_{rp} + c_2)}{(\mu_r^2 + \mu_p^2 + c_1)(\sigma_r^2 + \sigma_p^2 + c_2)}$$

Dieses Maß kann besser Bilddifferenzen im Bezug zur menschlichen visuellen Wahrnehmung bewerten. Es orientiert sich mehr an strukturelle Informationen und Helligkeiten. Dabei ist SSIM zwischen 0 und 1 definiert. Identische Bilder haben einen Score von 1 [7].
Im Rahmen dieser Modularbeit ist die Klassifikationsmetrik *Accuracy* ebenfalls betrachtet worden. Obwohl das zu lösende Problem primär ein Regressionsproblem ist, kann Keras dieses Problem zu einem Klassifikationsproblem umstellen. Dabei muss das Modell den exakten Pixelwert treffen, damit es korrekt liegt. Ein kleiner Fehler wird damit genauso hart bestraft wie ein großer Fehler. Es ist anzunehmen, dass diese Metrik auch funktionieren kann. Da aber der *MSE* die gleiche Aufgabe besser erfüllt, wurde die *Accuracy* nicht weiter betrachtet. In Experimenten überzeugte der *MSE* trotz der Vorteile der oben genannten Metriken. Die anderen Metriken wurden zu Validierungszwecken weiterhin mitgeführt.

B. Optimierer und Lernrate

Für das Training wurden die Optimierer *Adam* und *SGD* mit *Nesterov momentum* verwendet. Andere Optimierer wie *RMSprop* und *Adagrad* wurden nicht betrachtet, da diese meist eine schlechtere Performance im Vergleich zu den betrachteten Verfahren liefern. Dies hat den Vorteil, dass die Anzahl der Hyperparameter reduziert wird. Die Lernrate ist von dem verwendeten Optimierer abhängig. *Adam* wurde mit einer Lernrate von 0,0005, β_1 mit 0,9, β_2 mit 0,999, epsilon mit $1e-08$ und einem Decay von $1e-5$ basierend auf beliebten Werten aus der Literatur getestet [8]. SGD wurde mit einer Lernrate von 0,001 und Momentum von 0,9 getestet. Von den beiden Verfahren lieferte *Adam* erheblich bessere Ergebnisse. SGD braucht trotz Momentum etwas mehr Epochen um denselben Loss wie *Adam* zu erreichen und hinkt diesem hinterher.

C. Mini Batch Size

Es war von Beginn an klar, dass *mini batch descent (MBGD)* verwendet wird. *Batch gradient descent (BGD)*, welches den ganzen Trainingsdatensatz auf einmal verwendet spielte von Beginn an keine Rolle, da es in den meisten Fällen schlechtere Ergebnisse liefert. Nun stellt sich die Frage, wie groß die Batchsize für MBGD sein soll. In der Literatur ist meist 32 der Standard. Im Rahmen dieser Hyperparameteroptimierung wurden die *Mini Batch Size* 2, 4, 8, 16 und 32 getestet. Größere Werte sind aufgrund des zu großen Speicherbedarfs nicht mehr möglich gewesen. *Stochastic Gradient Descent (SGD)*, das nur einen einzelnen Datenpunkt verwendet, wurde auch getestet. Die besten Ergebnisse liefert die *Mini Batch Size* 4.

VII. Finales Training

Zuletzt wird die beste Netzarchitektur mit dem besten Hyperparameter für 500 Epochen trainiert. Als Loss Funktion wird der *MSE* Loss verwendet. Der Optimierer ist *Adam* mit einer Lernrate von 0,0005. Die ausgewählte *Mini Batch Size* liegt bei 4.
Um Overfitting zu vermeiden, wird EarlyStopping verwendet, welches das Training automatisch beendet, sobald sich der *Loss* auf dem Validierungsdatensatz über 50 Iterationen nicht mehr verbessert. Die Lernkurve des finalen Trainings in Abbildung 4 sieht gut aus. Es ist kein Overfitting zu beobachten.

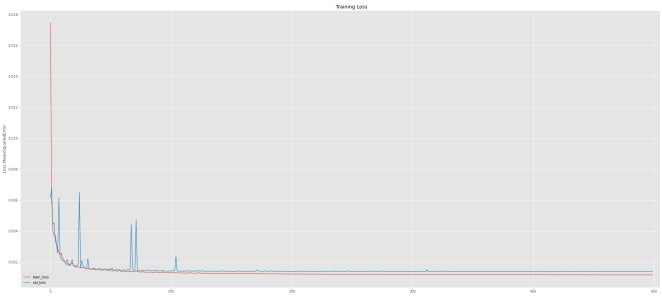

Abbildung 4: Lernkurve des finalen Trainings.

Das Netz erreichte zum Ende einen *MSE* von 0,0014 und einen *PSNR* von 28,44 auf dem Trainingsdatensatz.Auf dem Validierungsdatensatz wurde ein *MSE* von 0,0015 und ein *PSNR* von erreicht.

VIII. Resultat

Das Modell konnte auf dem Trainings- und Validierungsdatensatz sehr gute Ergebnisse erzielen. Die Tabelle 1 zeigt die Scores des finalen Modells nun auf dem Testdatensatz. Zusätzlich ist angegeben wie groß der Fehler des verrauschten Bildes zum Original ist und wie gut die OpenCV Funktion ”cv2.fastNlMeansDenoising()” das Bild rekonstruieren kann.
Die OpenCV Funktion scheitert an diesem sehr stark verrauschten Bildern gänzlich. Es verschlechtert die Bilder sogar. Der Autoencoder ist dagegen in der Lage das Rauschen gänzlich zu entfernen und verursacht im Durchschnitt nur wenige Bildfehler.

Metrik	Rauschen	OpenCV	Autoencoder
MSE	0,3127	310	0,0015
PSNR	9,8727	-1	28,127
SSIM	0,0363	0,0389	0,3922

Tabelle I: Fehler von Gaus Rauschen zu Original, OpenCV Rekonstruktion zu Original, Autoencoder Rekonstruktion zu Original

Diese recht guten Werte zeigen sich auch in den Bildern auf dem Testdatensatz in Abbildung 5. Wie bereits im Kapitel II beschrieben, kann der Autoencoder die Motorhaube und Karosserie mit Leichtigkeit rekonstruieren. Dies trifft ebenfalls auf die weißen Fahrspuren und Fensterrahmen zu. Allgemein werden Objekte die öfters in der Szene auftauchen besser rekonstruiert. Besonders gut rekonstruiert das Netz diese Bilder:

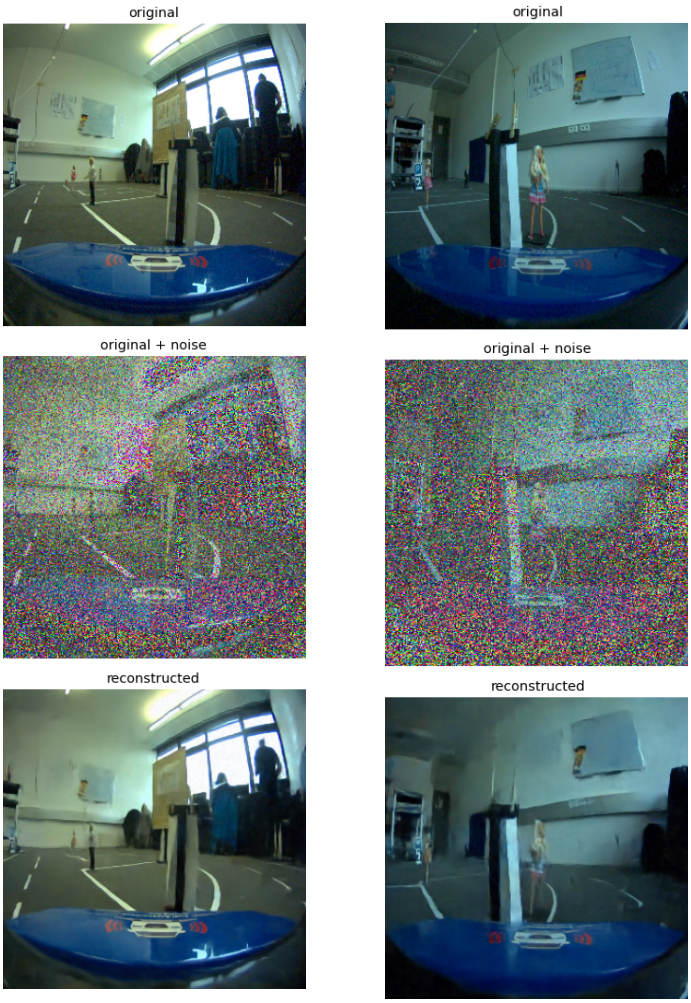

Abbildung 5: Zwei Beispielbilder aus dem Testdatensatz als Original, Verrauscht und durch den Autoencoder rekonstruiert

Schwierigkeiten hat das Modell bei sehr kleinen und schmalen Objekten, wie beispielsweise das Bein der Figur. Sehr wenigen Bildern enthalten nach der Rekonstruierung Bildfehler. Dies trifft vor allem auf Bilder zu, die sehr dunkle Bildbereiche beinhalten. Das Bild, welches das Modell am schlechtesten korrigieren konnte, ist das Bild in der Abbildung 6.

Abbildung 6: Bild mit Bildfehlern nach der Rekonstruierung durch den Autoencoder.

IX. Ausblick

Der trainierte Autoencoder war in der Lage, das ihm gestellte Problem sehr gut zu lösen. Nächste Schritte könnten sein, das Modell auf einem deutlich größeren und diverseren Datensatz zu trainieren.
Auch ist im Bereich der Modellarchitektur noch nicht alles ausgeschöpft. Weitere Ansätze wie zum Beispiel der in dem Paper “Generalized Denoising Auto-Encoders as Generative Models” [9] können zu weiteren Verbesserungen führen.

Literatur

[1] I. Goodfellow , Y. Bengio , and A. Courville, Deep Learning. MIT Press, 2016. Accessed: Jan. 24, 2022. [Online]. Available: http://www.deeplearningbook.org

[2] F. Chollet , “Building Autoencoders in Keras,” May 14, 2016. https://blog.keras.io/building-autoencoders-in-keras.html (accessed Jan. 22, 2022).

[3] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” arXiv.org, Dec. 10, 2015. https://arxiv.org/abs/1512.03385

[4] H. Patel, “Image Super-Resolution using Convolution Neural Networks and Auto-encoders,” Towards Data Science, Jun. 15, 2020. https://towardsdatascience.com/image-super-resolution-using-convolution-neural-networks-and-auto-encoders-28c9eceadf90 (accessed Jan. 23, 2022).

[5] J. Jordan, “Introduction to autoencoders.,” Jeremy Jordan, Mar. 19, 2018. Accessed: Jan. 24, 2022. [Online]. Available: https://www.jeremyjordan.me/autoencoders/

[6] Huynh-Thu and Ghanbari, “The accuracy of PSNR in predicting video quality for different video scenes and frame rates,” Telecommunication Systems, vol. 49, no. 1, pp. 35–48, Jun. 2010, doi: 10.1007/s11235-010-9351-x.

[7] Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, Ïmage quality assessment: from error visibility to structural similarity,ïn IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.

[8] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv.org, Dec. 22, 2014. https://arxiv.org/abs/1412.6980

[9] Y. Bengio, L. Yao, G. Alain, and P. Vincent, “Generalized Denoising Auto-Encoders as Generative Models,” arXiv.org, May 29, 2013. https://arxiv.org/abs/1305.6663