Deteksi Aksara Jawa Menggunakan AlexNet, VGGNet, dan ResNet

Faris Qanit, M. Aulia Arief, Nathan Arianto Wijaya, Rizki Fajar Aristanto, Yusuf Rachmad Walidain
Departemen Ilmu Komputer Dan Elektronika
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Gadjah Mada
{farisqanit, aulia.arief,rizkifajar456
nathanarianto, yusufrachmad}@ugm.ac.id

Abstrak

Aksara Jawa merupakan salah satu warisan budaya Indonesia yang banyak digunakan sejak pertengahan abad ke-15 dalam sastra maupun tulisan sehari - hari masyarakat Jawa. Namun, seiring dengan berkembangnya zaman, budaya yang seharusnya perlu dilestarikan ini mulai dilupakan dan juga tergantikan. Diperlukan suatu usaha untuk terus melestarikan aksara Jawa ini. Karena itu, sebagai usaha untuk meningkatkan kesadaran dan kemudahan dalam mempelajari budaya Jawa tersebut, pada penelitian ini akan dibuat sebuah sistem yang mampu mengenali gambar tulisan tangan aksara Jawa. Algoritma Convolutional Neural Network (CNN) digunakan sebagai basis pembuatan sistem.

1 Pendahuluan

Indonesia adalah negara kepulauan terbesar di dunia dengan 17.000 pulau. Berdasarkan Direktorat Jenderal Kependudukan dan Pencatatan Sipil (Dukcapil), tercatat jumlah penduduk Indonesia mencapai 275,36 juta jiwa pada Juni 2022 (DUKCAPIL, 2022). Keadaan ini menyebabkan Indonesia memiliki banyak budaya yang berbeda - beda, salah satunya adalah bahasa daerah.

Namun, seiring dengan berkembangnya zaman, budaya yang seharusnya perlu dilestarikan ini mulai dilupakan. Berbagai bahasa daerah yang ada mulai terancam punah. Salah satunya adalah bahasa Jawa terutama dalam penulisannya menggunakan aksara Jawa. Aksara Jawa aktif digunakan sejak pertengahan abad ke-15 dalam sastra maupun tulisan sehari - hari masyarakat Jawa. Dalam sejarahnya, aksara Jawa juga ditemukan dalam berbagai prasasti batu, lempeng logam, hingga naskah kertas pada perkembangannya (Wikipedia, 2022).

Penggunaan aksara jawa semakin menurun pada masa kini. Dibutuhkan suatu upaya untuk terus melestarikan penggunaan bahasa Jawa. Salah satu cara yang dapat dilakukan adalah melakukan digitalisasi terhadap aksara Jawa dan membuat sebuah sistem yang dapat mengenali tulisan aksara Jawa agar siapapun dapat mempelajari aksara Jawa. Pada penelitian ini akan dibuat sebuah sistem yang mampu mengenali gambar tulisan tangan aksara Jawa menggunakan beberapa arsitektur Convolutional Neural Network (CNN).

2 Landasan Teori

2.1 *Aksara Jawa*

Aksara Jawa terdiri dari 20 karakter dasar yang sering digunakan dalam penulisan. Bentuk karakter dasar dari Aksara Jawa dapat terlihat fig 1. Karakter-karakter tersebut memiliki pola yang hampir sama, tetapi masing-masing karakter memiliki citra yang berbeda.

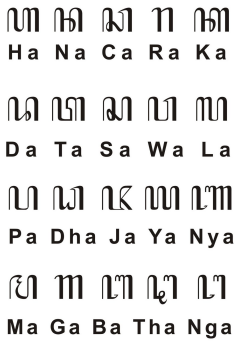

Figure 1: Karakter Aksara Jawa

2.2 *Deep Learning*

Deep Learning adalah sebuah cabang ilmu dari *machine learning* yang meniru jaringan saraf manusia. Pengolahan data dalam *deep learning* menggunakan sejumlah lapisan yang tersembunyi yang biasa disebut dengan *hidden layer*. Algoritma pada *deep learning* mampu mengekstraksi secara otomatis.

2.3 *Convolutional Neural Network*

Convolutional Neural Network adalah implementasi *deep learning* yang biasa digunakan untuk mendeteksi dan mengenali objek pada suatu gambar. Proses CNN umumnya melalui convolutional layer, ReLU layer, pooling layer, dan fully-connected layer yang dapat digambarkan pada fig 2.

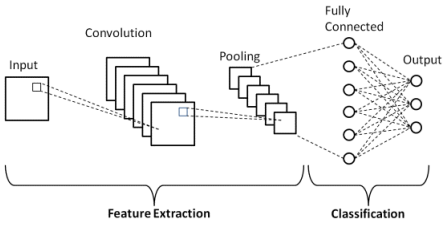

Figure 2: Arsitektur umum CNN

2.4 *AlexNet*

AlexNet merupakan sebuah arsitektur CNN yang terdiri dari lima *convolutional layers* dan *fully connected layers*, seperti yang dapat dilihat pada fig. 3. AlexNet menggunakan fungsi aktivasi ReLU, ketika saat itu *sigmoid* dan *tanh* adalah fungsi aktivasi yang biasanya digunakan. Penggunaan ReLU ini membantu mengatasi masalah *vanishing gradient*. Selain itu, untuk mengatasi *over-fitting* AlexNet juga menerapkan *data augmentation* dan *dropout layers* (Krizhevsky et al., 2012).

Layer Name		Filter Size	Stride	Padding	Number of Filters	Output Feature Map Size
Image Input Layer						120 × 240 × 3
Conv1	Conv. CCN Max Pooling	7 × 7 × 3 3 × 3	2 2	0 0	96	57 × 117 × 96 28 × 58 × 96
Conv2	Conv. CCN Max Pooling	5 × 5 × 96 3 × 3	1 2	2 0	128	28 × 58 × 128 13 × 28 × 128
Conv3	Conv.	3 × 3 × 128	1	1	256	13 × 28 × 256
Conv4	Conv.	3 × 3 × 256	1	1	256	13 × 28 × 256
Conv5	Conv. Max Pooling	3 × 3 × 256 3 × 3	1 2	1 0	128	13 × 28 × 128 6 × 13 × 128
Fully Connected Layers	Fc1 Fc2 Dropout Fc3 Softmax					4096 2048 116

Figure 3: Arsitektur AlexNet

2.5 *VGGNet*

VGGNet merupakan sebuah arsitektur CNN dimana seluruh *convolutional layers* berukuran 3x3 and *maxpool kernels* berukuran 2x2 dengan *stride* 2, seperti yang dapat dilihat pada fig. 4. Dibanding memiliki filter dengan ukuran besar, VGG menggunakan filter - filter kecil dengan jumlah yang lebih banyak. Ini dilakukan untuk menambah kedalaman *(depth) neural network* (Simonyan and Zisserman, 2015).

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 4: Arsitektur VGGNet

2.6 *ResNet*

ResNet merupakan sebuah arsitektur CNN yang memiliki konsep *residual blocks* yang digunakan untuk mengatasi permasalahan *vanishing/exploding gradient*. Pada ResNet juga dikenalkan dengan teknik *skip connections* yang diilustrasikan pada fig. 5. Sesuai dengan namanya, *skip connections* menyambungkan aktivasi lapisan ke lapisan selanjutnya dengan melewati beberapa lapisan yang berada di antara kedua lapisan tersebut (FG, 2022).

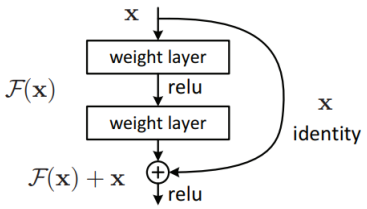

Figure 5: Konsep *residual block*

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 $\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 6: Arsitektur *residual network*

3 Metodologi

Pada penelitian ini, pengenalan huruf Jawa dimulai dengan mengambil dataset yang diperlukan dari Kaggle, kemudian, preprocessing dilakukan pada data. Setelah itu buat model yang akan digunakan pada klasifikasi, dimana pada penelitian ini akan digunakan arsitektur AlexNet, VGG11, dan ResNet18. Terakhir, *training* dan *testing* dilakukan pada model dengan dataset yang sudah disiapkan.

3.1 Dataset

Penelitian yang dilakukan menggunakan dataset yang berasal dari Kaggle. Dataset tersebut berisi sekitar 1500+ tulisan tangan digital dalam format JPG. Setiap huruf pada aksara jawa hanacaraka berisi sekitar 75 yang memiliki resolusi beragam tetapi dibawah 500x500 pixel.

Data yang ada kemudian dibagi menjadi tiga bagian, yaitu data training, data validation, dan data *testing*. Rasio pembagian yang digunakan adalah 69:16:15 persen dimana 69 untuk *training*, 16 untuk validation dan 15 untuk test. Sehingga data *training* yang akan digunakan sebanyak 1061 gambar, 266 gambar berupa validasi, dan sisanya 235 gambar merupakan data tes.

3.2 Preprosesing

Preprocessing dataset juga dilakukan dalam *training dataset* dengan melakukan *random rotation* dan *random crop*. Berikut adalah sampel acak dari training dataset terlihat pada fig 8

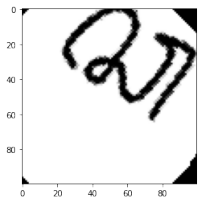

Figure 7: Sampel acak *training dataset*

3.3 Buat Model

Deklarasikan arsitektur atau model CNN yang akan digunakan pada penelitian. Pada penelitian ini akan dibandingkan arsitektur AlexNet, VGG11, dan arsitektur *Residual Network* dengan kedalaman 18 layer (ResNet18). Arsitektur ResNet18.

3.4 Training

Proses training adalah proses untuk melatih model yang telah dibuat sebelumnya dengan menggunakan sejumlah data training yang telah disiapkan. Fungsi optimisasi yang digunakan pada training yaitu *Adam Optimization Algorithm*. Kemudian, jenis *scheduler* yang dipakai adalah *One Cycle Learning Rate*, yaitu merupakan kombinasi dengan cara harus meningkatkan dan menurunkan secara opsional *learning rate* pada setengah siklus pertama, kemudian harus menurunkan dan meningkatkan secara opsional *learning rate* pada setengah siklus selanjutnya. Berikut merupakan ilustrasi dari nilai *learning rate* dengan menggunakan *One Cycle Learning Rate*.

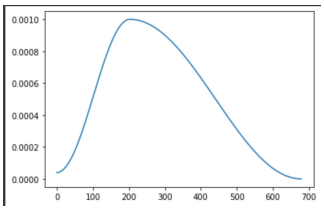

Figure 8: Nilai *learning rate* pada *One Cycle Learning Rate*

3.5 Testing

Proses testing adalah proses untuk menguji / mengevaluasi model yang telah di-trainingkan dengan menggunakan sejumlah data testing yang telah disiapkan.

4 Hasil dan Analisis

4.1 AlexNet

Pengujian dilakukan menggunakan arsitektur AlexNet, kemudian diperoleh akurasi validasi dan loss validasi sebagai berikut:

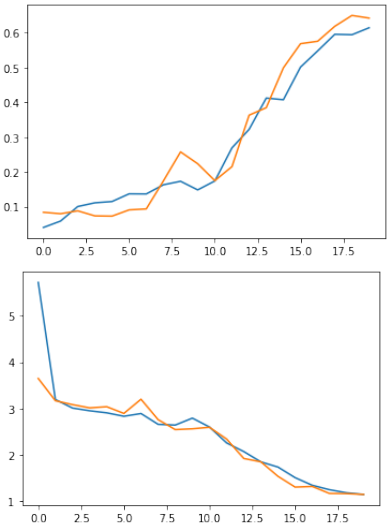

Figure 9: Hasil akurasi *training dataset* menggunakan AlexNet

	precision	recall	f1-score	support
0	0.90	0.75	0.82	12
1	0.59	0.83	0.69	12
2	1.00	0.33	0.50	12
3	0.00	0.00	0.00	11
4	0.75	0.25	0.38	12
5	0.44	0.92	0.59	12
6	1.00	0.83	0.91	12
7	0.90	0.75	0.82	12
8	1.00	0.42	0.59	12
9	0.79	0.92	0.85	12
10	0.44	0.67	0.53	12
11	0.90	0.75	0.82	12
12	0.88	0.58	0.70	12
13	0.36	0.45	0.40	11
14	0.91	0.91	0.91	11
15	0.43	0.50	0.46	12
16	0.50	0.45	0.48	11
17	0.69	0.82	0.75	11
18	0.41	0.75	0.53	12
19	0.77	0.83	0.80	12
accuracy			0.64	235
macro avg	0.68	0.64	0.63	235
weighted avg	0.69	0.64	0.63	235

Figure 10: Nilai akurasi AlexNet menggunakan 20 *epochs*

Hasil nilai akurasi model yang dilatih menggunakan VGG juga tinggi dengan nilai 0.64 atau 64% sesuai dengan fig. 10.

4.2 VGG11

Percobaan pada VGG 11 dilakukan menggunakan *pre-trained model* dari *library* PyTorch. Pada awalnya percobaan menggunakan arsitektur VGG 11 dilatih dengan *hardware* CPU. Pelatihan model menggunakan CPU memakan waktu kurang lebih 2 jam dan model sangat *overfit* disebabkan data yang digunakan adalah data dalam bentuk *image*. Kemudian dengan menghilangkan kustomisasi *fully-connected layer* pada percobaan sebelumnya dan mengganti *hardware* menjadi GPU, diperoleh grafik *accuracy* dan *loss* pada data *training* dan *testing* sebagai berikut.

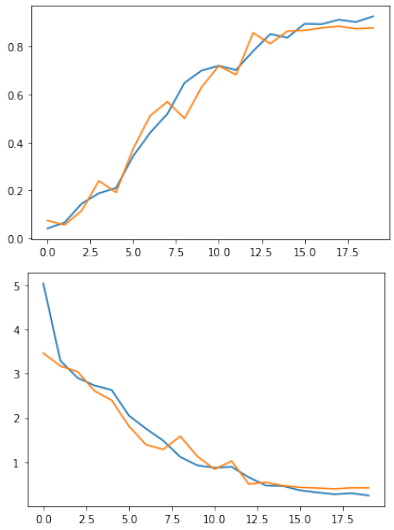

Figure 11: Hasil akurasi *training dataset* menggunakan VGG 11

	precision	recall	f1-score	support
0	0.69	0.92	0.79	12
1	1.00	1.00	1.00	12
2	1.00	0.83	0.91	12
3	1.00	1.00	1.00	11
4	0.73	0.67	0.70	12
5	0.85	0.92	0.88	12
6	1.00	1.00	1.00	12
7	1.00	0.83	0.91	12
8	0.89	0.67	0.76	12
9	1.00	0.92	0.96	12
10	0.86	1.00	0.92	12
11	0.92	0.92	0.92	12
12	1.00	0.58	0.74	12
13	0.65	1.00	0.79	11
14	1.00	0.91	0.95	11
15	1.00	1.00	1.00	12
16	1.00	1.00	1.00	11
17	0.92	1.00	0.96	11
18	1.00	1.00	1.00	12
19	0.69	0.75	0.72	12
accuracy			0.89	235
macro avg	0.91	0.90	0.89	235
weighted avg	0.91	0.89	0.89	235

Figure 12: Nilai akurasi VGG 11 menggunakan 20 *epochs*

Fig. 11 menunjukkan bahwa pemakaian arsitektur VGG11 menghasilkan model yang baik karena model tidak mengalami *overfitting*. Hasil nilai akurasi model yang dilatih menggunakan VGG juga tinggi dengan nilai 0.89 atau 89% sesuai dengan fig. 12. Hasil ini merupakan hasil yang jauh lebih baik dari percobaan dengan AlexNet.

4.3 ResNet18

Pengujian terkahir dilakukan menggunakan arsitektur ResNet 18 tanpa menambahkan parameter tambahan pada arsitektur kemudian diperoleh akurasi validasi dan loss validasi sebagai berikut.

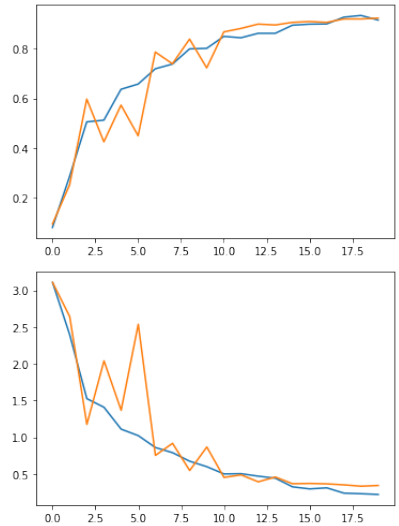

Figure 13: Hasil akurasi *training dataset* menggunakan ResNet18

Grafik pada gambar 13 menunjukkan bahwa model yang dilatih tidak mengalami *overfitting* karena antara *validation* dan *training* saling

berhimpitan. Model tersebut diuji dengan menggunakan *epoch* sebanyak 20. Hasil akurasi dari pengujian tersebut mendapatkan hasil tertinggi dari ketiga arsitektur yang dicoba dengan nilai akurasi 0.94 atau 94% yang dapat dilihat pada gambar 14.

	precision	recall	f1-score	support
0	0.92	1.00	0.96	12
1	0.92	1.00	0.96	12
2	0.91	0.83	0.87	12
3	1.00	0.91	0.95	11
4	1.00	0.75	0.86	12
5	0.92	0.92	0.92	12
6	1.00	1.00	1.00	12
7	1.00	1.00	1.00	12
8	1.00	0.83	0.91	12
9	1.00	0.92	0.96	12
10	0.85	0.92	0.88	12
11	0.92	0.92	0.92	12
12	1.00	0.92	0.96	12
13	0.92	1.00	0.96	11
14	1.00	0.91	0.95	11
15	1.00	1.00	1.00	12
16	1.00	1.00	1.00	11
17	0.91	0.91	0.91	11
18	0.86	1.00	0.92	12
19	0.75	1.00	0.86	12
accuracy			0.94	235
macro avg	0.94	0.94	0.94	235
weighted avg	0.94	0.94	0.94	235

Figure 14: Nilai akurasi ResNet18 menggunakan 20 *epochs*

5 Kesimpulan dan Saran

5.1 Kesimpulan

Deteksi dan klasifikasi aksara Jawa adalah suata cara dalam mempelajari aksara jawa karena sistem dapat mendeteksi tulisan aksara jawa dengan tingkat akurasi yang cukup tinggi. Penelitian yang telah dilakukan menggunakan model CNN berupa AlexNet, VGG11, dan ResNet18. Penggunaan tiga arsitektur dilakukan untuk melihat bagaimana input dataset berpengaruh tingkat akurasi data dalam jumlah kecil. Dari hasil percobaan, dapat disimpulkan bahwa model ResNet18 lebih baik dengan akurasi secara keseluruhan dari 235 gambar test, yaitu 94%. Maka dari itu, model ini sangat dianjurkan untuk diimplementasikan dalam sistem pendeteksi tulisan aksara untuk membantu pengguna mengetahui jenis tulisan aksara Jawa yang di analisa.

Penelitian ini juga membuktikan bahwa *deep CNN models* yang digunakan pada eksperimen ini, bekerja dengan baik pada dataset berukuran kecil, terutama pada ResNet. Jadi, selama modifikasi pada hyperparameter dilakukan dengan baik, serta digunakannya *data augmentation* dan *dropout*, terbukti bahwa deep model bekerja pada dataset kecil.

5.2 Saran

Untuk penelitian atau pengembangan selanjutnya, ukuran dataset dapat diperbesar untuk melihat apakah prediksi akan berjalan dengan baik pada variasi data yang lebih banyak. Selain itu, arsitektur CNN lainnya juga dapat dicoba diterapkan, misalnya seperti Inception, DenseNet, dan LeNet untuk melihat apakah akurasi yang didapatkan akan lebih baik atau tidak pada dataset kecil. *Deep models* lain seperti RCNN (*Region Based CNN*) dan YOLO (*You Only Look Once*) juga bisa menjadi bahan penelitian lainnya.

References

DUKCAPIL. 2022. Dukcapil Kemendagri Rilis Data Penduduk Semester I Tahun 2022, Naik 0,54% Dalam Waktu 6 Bulan.

Pawang FG. 2022. Residual networks (resnet) – deep learning.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. ArXiv:1409.1556 [cs].

Wikipedia. 2022. Aksara Jawa.