Int. j. adv. multidisc. res. stud. 2024; 4(4):982-989

Received: 01-07-2024
Accepted: 10-08-2024

International Journal of Advanced Multidisciplinary Research and Studies

ISSN: 2583-049X

Head Classification based on Convolutional Neural Network

[1] Panca Mudjirahardjo, [2] Aqil Gama Rahmansyah, [3] Alya Shafa Dianti

[1, 2] Department of Electrical Engineering, Faculty of Engineering, Universitas Brawijaya, Malang-Indonesia
[3] Department of Statistics, Faculty of Mathematic and Natural Science, Universitas Brawijaya, Malang-Indonesia

Corresponding Author: **Panca Mudjirahardjo**

Abstract
In this paper, we study the head classification using Convolutional Neural Network (CNN). We study the effect of optimizer in various network architecture. We use INRIA datasets and python programming language. We study and evaluate 2 models and 8 optimizers. We evaluate the validation accuracy and training computation time in one epoch. In our experiment result, except optimizers of SGD and Adadelta, the validation accuracy are good. Their performance are above 90%. The average training time of 1 epoch is 3 second.

Keywords: Head Classification, CNN, Optimizer, Network Architecture, Validation Accuracy

1. Introduction

There are many object classification method implemented in automation system. However, nowadays, CNN become a popular architecture in object classification. This due to the feature extraction and classifier task to be done in sequential process. Convolutional stage is a feature extraction, followed by a neural network as an object classifier.

Object classification is a task in computer vision where the goal is to identify and categorize objects within an image. This process involves determining the presence and class of objects from a set of predefined categories. Here's a breakdown of how object classification works and the approaches used:

1. Overview of Object Classification

1. **Objective:** The main goal is to assign a label or category to an object in an image based on its visual content. For example, classifying an image as "cat," "dog," or "car."
2. **Applications:** Object classification is widely used in various fields such as autonomous driving (detecting pedestrians or other vehicles), medical imaging (identifying tumors), and image search engines (finding similar images).

2. Process

1. Data Collection and Annotation:
- **Dataset:** Collect a large dataset of images where each image is labeled with the correct category. This dataset is often divided into training, validation, and test sets.
- **Annotation:** Label each image with the correct class. This can be done manually or using automated tools.

2. Preprocessing:
- **Resizing:** Standardize image sizes to ensure uniform input dimensions for the model.
- **Normalization:** Scale pixel values to a range (e.g., 0 to 1) to improve model convergence.
- **Augmentation:** Apply transformations like rotations, flips, and shifts to increase dataset diversity and robustness.

3. Model Training:
- **Feature Extraction:** Use models like Convolutional Neural Networks (CNNs) to automatically extract features from images.
- **Classification Layer:** After feature extraction, a classifier (often a fully connected layer or a softmax layer) assigns probabilities to each class.
- **Loss Function:** Use functions like Cross-Entropy Loss to measure how well the model's predictions match the actual labels.

- **Optimization:** Adjust model parameters using optimization algorithms like SGD or Adam to minimize the loss function.

4. Evaluation:

- **Metrics:** Evaluate the model's performance using metrics such as accuracy, precision, recall, and F1 score. For multi-class problems, confusion matrices can also be helpful.

5. Inference:

- **Prediction:** For new, unseen images, the trained model predicts the class based on learned features.

3. Techniques and Models

1. Traditional Methods:

- **Handcrafted Features:** Early approaches used manually designed features (e.g., HOG, SIFT) combined with classifiers like Support Vector Machines (SVMs). These methods are less effective than modern deep learning techniques but were widely used before CNNs became dominant.

2. Deep Learning Approaches:

- **Convolutional Neural Networks (CNNs):** CNNs are the backbone of modern object classification. They consist of convolutional layers, activation functions, pooling layers, and fully connected layers to learn hierarchical features.
- **LeNet-5:** One of the earliest CNN architectures designed for digit recognition.
- **AlexNet:** A deep CNN that won the 2012 ImageNet competition and demonstrated the power of deep learning.
- **VGG:** Known for its simplicity and depth, with architectures like VGG16 and VGG19.
- **ResNet:** Introduces residual connections to allow very deep networks (e.g., ResNet50, ResNet101).
- **Inception:** Features modules with multiple filter sizes to capture various aspects of the data.

3. Transfer Learning:

- **Pre-trained Models:** Use models pre-trained on large datasets like ImageNet and fine-tune them on specific tasks. This approach leverages previously learned features and speeds up the training process.
- **Feature Extraction:** Use a pre-trained model to extract features from images and train a simple classifier on top of these features.

4. Advanced Techniques:

- **Attention Mechanisms:** Improve classification by focusing on important parts of the image, useful in tasks where object localization and context are crucial.
- **Ensemble Methods:** Combine multiple models to improve classification performance by aggregating their predictions.

4. Challenges

1. **Variability:** Objects can vary in size, shape, color, and orientation, making classification difficult.
2. **Class Imbalance:** Some classes may have fewer examples, leading to biased models.
3. **Adversarial Attacks:** Small perturbations to input images can fool models into making incorrect predictions.
4. **Computational Resources:** Training deep networks requires significant computational power and memory.

Object classification continues to be an evolving field with ongoing research aimed at improving accuracy, efficiency, and robustness of models.

As we know, **feature extraction** is a crucial step in data preprocessing, particularly in fields like machine learning, computer vision, and signal processing. It involves identifying and selecting the most relevant information (features) from raw data to improve the efficiency and performance of models. Here's a brief overview of feature extraction. The main goal is to reduce the dimensionality of the data while preserving important information. This makes it easier and faster to train models, and often leads to better performance.

Some techniques for feature extraction, are:

- **Statistical Methods:** Techniques like mean, variance, skewness, and kurtosis that summarize the statistical properties of the data.
- **Transformation Methods:** Methods such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) that transform the data into a new space where features are uncorrelated or optimized for classification.
- **Domain-Specific Methods:** In image processing, for instance, techniques like edge detection, histograms of oriented gradients (HOG), or convolutional neural network (CNN) features are used. In text processing, methods like term frequency-inverse document frequency (TF-IDF) or word embeddings (e.g., Word2Vec) are common.

Feature extraction is a blend of art and science, often requiring domain knowledge to determine which features will be most useful for a given problem.

A **Convolutional Neural Network** (CNN) is a specialized type of artificial neural network designed to process structured grid data, such as images. CNNs are particularly powerful for tasks related to image recognition and computer vision but have also been applied successfully to other types of data.

Here's a high-level overview of how CNNs work and their key components:

1. Architecture,

- **Convolutional Layers:** These layers apply convolutional filters (kernels) to the input data to create feature maps. Each filter detects specific features such as edges or textures. The convolution operation involves sliding the filter over the input and computing the dot product.
- **Activation Functions:** After the convolution operation, the feature maps are passed through an activation function, typically the Rectified Linear Unit (ReLU). ReLU introduces non-linearity by setting all negative values to zero, allowing the network to learn complex patterns.
- **Pooling Layers:** Pooling (or subsampling) layers reduce the spatial dimensions of the feature maps, which helps in reducing computation and preventing overfitting. Common pooling operations include Max Pooling (taking the maximum value in a region) and Average Pooling (taking the average value).
- **Fully Connected Layers:** After several convolutional and pooling layers, the resulting feature maps are flattened into a vector and passed through fully connected (dense) layers. These layers perform the final

classification or regression tasks based on the extracted features.

- **Dropout Layers:** Dropout is a regularization technique used to prevent overfitting. During training, dropout layers randomly set a fraction of the neurons to zero, which helps the network to generalize better.

2. Training Process,

- **Forward Propagation:** During training, an input image is passed through the network layer by layer, and the output is computed. Each layer applies its specific operation (convolution, activation, pooling, etc.) to transform the input data into higher-level features.
- **Loss Function:** The output of the network is compared to the true labels using a loss function (e.g., Cross-Entropy Loss for classification tasks). The loss function quantifies how well the network is performing.
- **Backpropagation:** The network's weights are updated to minimize the loss function. Backpropagation computes the gradients of the loss with respect to each weight using the chain rule and adjusts the weights accordingly. This process typically uses optimization algorithms like Stochastic Gradient Descent (SGD) or Adam.

3. Advantages,

- **Feature Learning:** CNNs automatically learn features from the data rather than relying on manual feature extraction. This makes them highly effective for tasks like image and speech recognition.
- **Spatial Hierarchy:** CNNs capture spatial hierarchies and local patterns, which are crucial for understanding images and other grid-like data.
- **Parameter Sharing:** Convolutional layers use shared weights (filters) across different parts of the input, which reduces the number of parameters compared to fully connected networks and makes them more computationally efficient.

4. Variants and Extensions,

- **Deep CNNs:** Networks with many convolutional layers, such as VGG, ResNet, and Inception, which can learn more complex features and achieve higher performance on challenging tasks.
- **Transfer Learning:** Using pre-trained CNNs (e.g., trained on ImageNet) and fine-tuning them for specific tasks can leverage existing knowledge and improve performance with less training data.
- **Object Detection:** Specialized architectures like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector) extend CNNs for detecting objects within images and localizing them with bounding boxes.
- **Segmentation:** Networks like U-Net and Mask R-CNN are used for pixel-wise segmentation tasks, such as identifying regions of interest within images.

CNNs have revolutionized many fields by providing state-of-the-art performance on tasks that involve spatial data, and they continue to be an active area of research and development.

In machine learning, an **optimizer** is an algorithm or method used to adjust the parameters (weights) of a model in order to minimize the loss function, which measures how well the model's predictions match the actual target values. The choice of optimizer can significantly impact the performance and efficiency of the training process. Here's an overview of various optimizers and their roles:

1. Stochastic Gradient Descent (SGD) is a popular optimization algorithm used in machine learning and deep learning for training models. It is an iteration-based variant of gradient descent that updates model parameters more frequently, which can help to speed up the training process.
2. The Adam optimizer (short for **Adaptive Moment Estimation**) is a widely used optimization algorithm in machine learning and deep learning that combines the advantages of two other popular optimizers: **Momentum** and **RMSprop**. Adam is designed to be computationally efficient, have low memory requirements, and be well-suited for problems with large datasets and parameters.
3. The **RMSprop (Root Mean Square Propagation)** optimizer is an adaptive learning rate optimization algorithm designed to address some of the challenges associated with traditional gradient descent methods. It is particularly useful in training neural networks and handling problems with noisy gradients or varying gradient scales.
4. The **Adagrad (Adaptive Gradient Algorithm)** optimizer is an adaptive learning rate optimization algorithm that adjusts the learning rate for each parameter individually based on the historical gradients. This makes it particularly effective for problems with sparse gradients or features, such as those encountered in natural language processing and computer vision tasks.
5. The **AdamW (Adam with Weight Decay)** optimizer is an extension of the Adam optimizer that decouples weight decay from the optimization steps, leading to improved regularization and generalization. It addresses a common issue in Adam where weight decay (used for regularization) is combined with the gradient updates in a way that can affect the performance of the model.
6. The **Adadelta** optimizer is an extension of the Adagrad optimizer designed to address some of the limitations associated with Adagrad. Adagrad's primary issue is its aggressive, monotonically decreasing learning rate, which can lead to very slow convergence or early stopping. Adadelta aims to mitigate this problem by using a moving window of accumulated past gradients to adapt learning rates and maintain more effective updates over time.
7. The **Adamax** optimizer is a variant of the Adam optimizer, designed to handle certain limitations of Adam and provide additional robustness. It is particularly useful for training deep neural networks and works well in scenarios where the gradients are very sparse or the data is noisy.
8. The **Nadam (Nesterov-accelerated Adaptive Moment Estimation)** optimizer is a combination of two optimization techniques: Nesterov Accelerated Gradient (NAG) and the Adam optimizer. It merges the benefits of both methods to improve training stability and convergence in deep learning models.

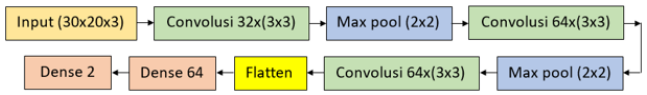
Choosing the right architecture and optimizer will improve the classification accuracy.

2. Method

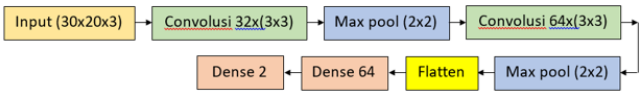
Our study method is depicted in Fig 1. In that figure, the input image size is 30×20 in color format, as shown in Fig 2.

Convolution and max pooling stages are to extract image's feature.

The first convolution uses 32 kernels of size 3×3, followed max pooling of size 2×2, followed the second convolution uses 64 kernels of size 3×3, and so on. After flatten, i.e. converting 2D array into 1D array, we use 64 nodes in hidden layer. Finally we use 2 outputs layer as 2 classes.



(a)



(b)

Fig 1: Architectures we used in this study (a) model-1 (b) model-2

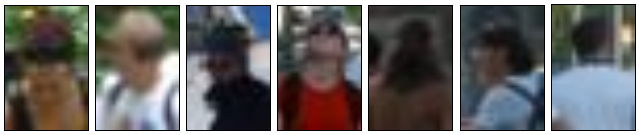


Fig 2: Some of head in INRIA datasets [2]

In Fig 2, the head data captured in various position and pose.

3. Experimental Result

We perform the experiment to classify two classes, head and no-head class. As images in Fig 2 show some of head data for head class and another images are no-head class. We have 4000 image files belonging to 2 classes, using 3000 files for training and 1000 files for validation.

We put the images in directory image:

D:\Dataset\

Head_OK\

head_ok(1).png
head_ok(2).png
head_ok(3).png

Head_NG\

head_ng(1).png
head_ng(2).png
head_ng(3).png

We use python language to implement this experiment.

The programming code to create the architecture of model-1 and model-2, as below:

```
def model_1():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(30, 20, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model

def model_2():
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(30, 20, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(num_classes))

    print('')
    model.summary()
    return model
```

Model summary of model-1 and model-2 are depicted in Fig 3 and Fig 4 respectively.

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 9, 32)	0
conv2d_1 (Conv2D)	(None, 12, 7, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 64)	0
conv2d_2 (Conv2D)	(None, 4, 1, 64)	36,928
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 64)	16,448
dense_1 (Dense)	(None, 2)	130

Total params: 72,898 (284.76 KB)
Trainable params: 72,898 (284.76 KB)
Non-trainable params: 0 (0.00 B)

Fig 3: Model summary of model-1

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 18, 32)	896
max_pooling2d (MaxPooling2D)	(None, 14, 9, 32)	0
conv2d_1 (Conv2D)	(None, 12, 7, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 3, 64)	0
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 64)	73,792
dense_1 (Dense)	(None, 2)	130

Total params: 93,314 (364.51 KB)
Trainable params: 93,314 (364.51 KB)
Non-trainable params: 0 (0.00 B)

Fig 4: Model summary of model-2

The complete source code are written in Program-1.

Program-1:

```
print('CNN, part 3 ..\n'*5)
print("ARZETI_Getsuyoubi,12.08.2024; 03:43")
print("Panca" +
      " Mudjirahardjo")
print("")
print("==================================")

print("")

modelKE = input('What model (1,2) : ')

print('')
optim = input('Optimizer: (1)SGD, (2)ADAM, (3)RMSprop, (4)Adagrad, (5)AdamW, (6)Adadelta, (7)Adamax, (8)Nadam : ')
if optim=='1':
   optim='SGD'
   print('-- optimizer: SGD')
elif optim=='2':
   optim='ADAM'
   print('-- optimizer: ADAM')
elif optim=='3':
   optim='RMSprop'
   print('-- optimizer: RMSprop')
elif optim=='4':
   optim='Adagrad'
   print('-- optimizer: Adagrad')
elif optim=='5':
   optim='AdamW'
   print('-- optimizer: AdamW')
elif optim=='6':
   optim='Adadelta'
   print('-- optimizer: Adadelta')
elif optim=='7':
   optim='Adamax'
   print('-- optimizer: Adamax')
elif optim=='8':
   optim='Nadam'
   print('-- optimizer: Nadam')

print('')
ep = input('Jumlah epoch (1 epoch 42 sec !): ')
ep = int(ep)

# ----------------------------------------------------------------------

import numpy as np
import os
os.environ['TF_ENABLE_ONEDNN_OPTS'] = '0'

from PIL import Image
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from keras.preprocessing import image

# ----------------------------------------------------------------------

def model_1():
   model = models.Sequential()
      model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(30, 20, 3)))
   model.add(layers.MaxPooling2D((2, 2)))
   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
   model.add(layers.MaxPooling2D((2, 2)))
   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
   model.add(layers.Flatten())
   model.add(layers.Dense(64, activation='relu'))
   model.add(layers.Dense(num_classes))

   print('')
   model.summary()
   return model

def model_2():
   model = models.Sequential()
      model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(30, 20, 3)))
   model.add(layers.MaxPooling2D((2, 2)))
   model.add(layers.Conv2D(64, (3, 3), activation='relu'))
   model.add(layers.MaxPooling2D((2, 2)))
   model.add(layers.Flatten())
   model.add(layers.Dense(64, activation='relu'))
   model.add(layers.Dense(num_classes))

   print('')
   model.summary()
   return model

# ----------------------------------------------------------------------

data_dir = "D:\Datasets"

train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.25,
  subset="training",
  seed=123,
  image_size=(30, 20),
  batch_size=20)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.25,
  subset="validation",
  seed=123,
  image_size=(30, 20),
  batch_size=20)

class_names = train_ds.class_names
print(class_names)

import matplotlib.pyplot as plt

for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break

normalization_layer = tf.keras.layers.Rescaling(1./255)

normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
```

```
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))

AUTOTUNE = tf.data.AUTOTUNE

train_ds =
train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)

num_classes = 2

# ------------------------------------------------------------------

print("")
print("")
if modelKE == '1':
    model = model_1()
    print("")
    print('---- model ke 1 (satu) ---')
elif modelKE == '2':
    model = model_2()
    print("")
    print('---- model ke 2 (dua) ---')

print('')
print('---- Optimizer: ' +optim)
print('---- training dimulai --- ')
print('')

model.compile(
  optimizer=optim,
   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_l
ogits=True),
  metrics=['accuracy'])

history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=ep
)

plt.figure('Model: ' +modelKE +' , optimizer: ' +optim)
plt.plot(history.history['accuracy'], label='train_accuracy')
plt.plot(history.history['val_accuracy'], label =
'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
plt.show()

print('')
print(' ----- model evaluate ---- ')
test_loss, test_acc = model.evaluate(val_ds)
```

A. Model-1 Result

The training and validation accuracy of model-1 with various optimizers are shown in Fig 5.

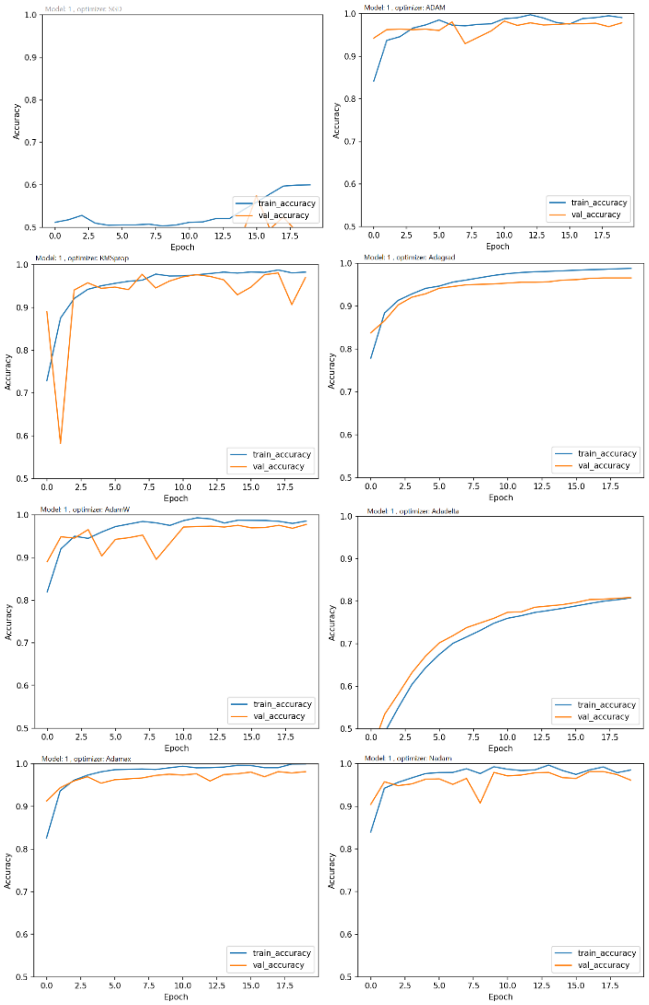
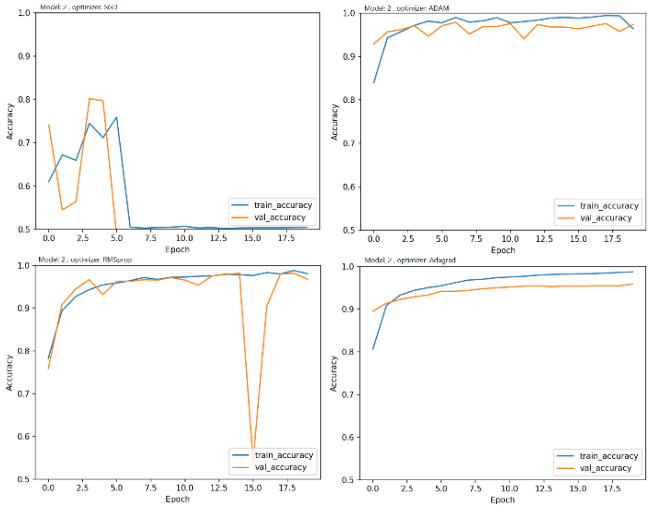


Fig 5: Training and validation accuracy of model-1 with various optimizers

B. Model-2 Result

The training and validation accuracy of model-2 with various optimizers are shown in Fig 6.



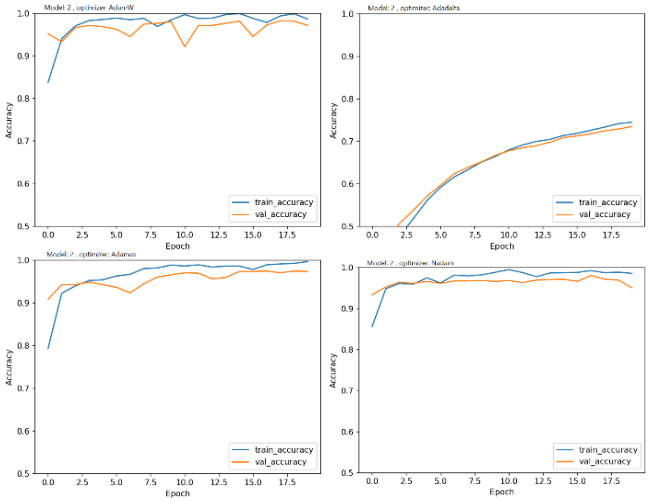


Fig 6: Training and validation accuracy of model-2 with various optimizers

```
---- model ke 2 (dua) ---

---- Optimizer: Adamax
---- training dimulai ---

Epoch 1/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 8s 35ms/step - accuracy: 0.6875 - loss: 4.7028 - val_accuracy: 0.9090 - val_loss: 0.2569
Epoch 2/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9152 - loss: 0.2569 - val_accuracy: 0.9430 - val_loss: 0.1711
Epoch 3/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 20ms/step - accuracy: 0.9398 - loss: 0.1849 - val_accuracy: 0.9430 - val_loss: 0.1806
Epoch 4/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9536 - loss: 0.1456 - val_accuracy: 0.9490 - val_loss: 0.1773
Epoch 5/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9571 - loss: 0.1228 - val_accuracy: 0.9430 - val_loss: 0.1820
Epoch 6/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 4s 27ms/step - accuracy: 0.9697 - loss: 0.0884 - val_accuracy: 0.9370 - val_loss: 0.2279
Epoch 7/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 23ms/step - accuracy: 0.9721 - loss: 0.0757 - val_accuracy: 0.9240 - val_loss: 0.2845
Epoch 8/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - accuracy: 0.9828 - loss: 0.0603 - val_accuracy: 0.9450 - val_loss: 0.1891
Epoch 9/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 22ms/step - accuracy: 0.9835 - loss: 0.0514 - val_accuracy: 0.9610 - val_loss: 0.1252
Epoch 10/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 4s 25ms/step - accuracy: 0.9881 - loss: 0.0373 - val_accuracy: 0.9660 - val_loss: 0.0944
Epoch 11/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 4s 26ms/step - accuracy: 0.9875 - loss: 0.0372 - val_accuracy: 0.9710 - val_loss: 0.0835
Epoch 12/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9883 - loss: 0.0372 - val_accuracy: 0.9700 - val_loss: 0.0856
Epoch 13/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9847 - loss: 0.0466 - val_accuracy: 0.9570 - val_loss: 0.1554
Epoch 14/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9865 - loss: 0.0391 - val_accuracy: 0.9600 - val_loss: 0.1572
Epoch 15/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9856 - loss: 0.0417 - val_accuracy: 0.9740 - val_loss: 0.0834
Epoch 16/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9783 - loss: 0.0664 - val_accuracy: 0.9740 - val_loss: 0.0758
Epoch 17/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 18ms/step - accuracy: 0.9882 - loss: 0.0323 - val_accuracy: 0.9750 - val_loss: 0.0732
Epoch 18/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 3s 17ms/step - accuracy: 0.9937 - loss: 0.0210 - val_accuracy: 0.9710 - val_loss: 0.0679
Epoch 19/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 0.9750 - val_loss: 0.0603
Epoch 20/20
150/150 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.9992 - loss: 0.0117 - val_accuracy: 0.9740 - val_loss: 0.0779

 ----- model evaluate ----
50/50 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - accuracy: 0.9723 - loss: 0.0915
```

Fig 7: Training history of model-2, optimizer: adamax

The overall result of our study is depicted in Fig 8.

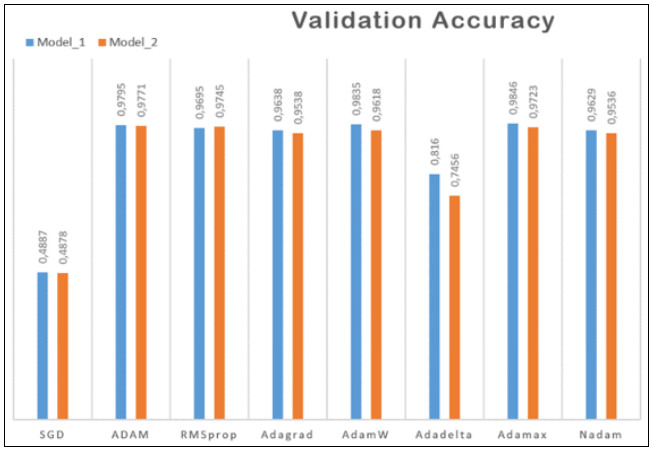


Fig 8: Comparison validation accuracy with various optimizers

4. Conclusions

We have studied and evaluated head classification for 2 models and 8 optimizers, using INRIA datasets. In our experiment result, except optimizers of SGD and Adadelta, the validation accuracy are good. Their performance are above 90% as shown in Fig 8. The average training time of 1 epoch is 3 second.

5. References

1. Alzubaidi L, Zhang J, Humaidi AJ, *et al*. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. Journal of Big Data, 2021.
2. Dalal N, Triggs B. Histograms of oriented gradients for human detection. 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), San Diego, CA, USA. 2005; 1:886-893. Doi: 10.1109/CVPR.
3. Yamashita R, Nishio M, Do RKG, *et al*. Convolutional neural networks: An overview and application in radiology. Insights Imaging. 2018; 9:611-629. Doi: https://doi.org/10.1007/s13244-018-0639-9
4. Will Cukierski. CIFAR-10 - Object Recognition in Images. Kaggle, 2013. https://kaggle.com/competitions/cifar-10
5. Kingma Diederik, Ba Jimmy. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations, 2014.
6. Gower RM, Loizou N, Qian X, Sailanbayev A, Shulgin E, Richtárik P. SGD: General analysis and improved rates. In international conference on machine learning (pp. 5200-5209). PMLR, 2019.
7. Mudjirahardjo P. The comparison of convolution and Max Pooling Process in real-time. International Journal of Advanced Multidisciplinary Research and Studies (IJAMRS). 2024; 4(3):1039-1044. ISSN: 2583-049x.
8. Mudjirahardjo P. Real-Time 2-D Convolution Layer for Feature Extraction. International Journal of Modern Engineering Research (IJMER). 2024; 14(03):211-216. ISSN: 2249-6645.
9. Mudjirahardjo P. Real-Time 2D Convolution and Max Pooling Process. International Journal of Computer Applications Technology and Research (IJCATR). 2024; 13(06):18-23. ISSN: 2319-8656. DOI: 10.7753/IJCATR1306.1003.
10. Mudjirahardjo P. The Effect of Grayscale, CLAHE Image and Filter Images in Convolution Process. International Journal of Advanced Multidisciplinary Research and Studies (IJAMRS). 2024; 4(3):943-946. ISSN: 2583-049x.
11. Mudjirahardjo P, Rahmansyah AG, Dianti AS. The Performance of Convolutional Neural Network Architecture in Classification. International Journal of Computer Applications Technology and Research (IJCATR). 2024; 13(08):115-122. ISSN: 2319-8656. Doi: 10.7753/IJCATR1308.1011.
12. Elshamy Reham, Abu Elnasr, Osama, Elhoseny Mohamed, Elmougy Samir. Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning. Scientific Reports. 2023; 13. Doi: 10.1038/s41598-023-35663-x.
13. Hassan E, Shams MY, Hikal NA, *et al*. The effect of choosing optimizer algorithms to improve computer vision tasks: A comparative study. Multimed Tools Appl. 2023; 82:16591-16633. Doi: https://doi.org/10.1007/s11042-022-13820-0

14. Abdulkadirov R, Lyakhov P, Nagornov N. Survey of Optimization Algorithms in Modern Neural Networks. Mathematics. 2023; 11:2466. Doi: https://doi.org/10.3390/math11112466