EDRAAK
Vol. **(2025)**, 2025, **pp**. 38–46
ISSN: 3078-8412

Research Article

Application of Deep Learning and Transfer Learning Techniques for Medical Image Classification

Tam Sakirin [1,*], Rachid Ben Said [2]

[1] *Information Communication Technology Department, University of Puthisastra, Phnom Penh, Cambodia*
[2] *Department of Computer Engineering, Ankara University, Ankara, Turkey*

ARTICLE INFO

Article History
Received 1 Nov 2024
Revised: 20 Des 2024
Accepted 20 Jan 2025
Published 12 Feb 2025

Keywords
Deep Learning (DL),
Transfer Learning (TL),
Medical Image Classification,
Convolutional Neural Networks (CNNs),
Feature Extraction.

ABSTRACT

The advancements in deep learning (DL) and transfer learning (TL) have transformed artificial intelligence, particularly in image classification. This research examines the theoretical foundations of DL and TL, focusing on their applications in medical image classification, specifically distinguishing between COVID-19, viral pneumonia, and normal lung conditions. By leveraging GPU-enabled high-performance computing and large labeled datasets, DL models particularly Convolutional Neural Networks (CNNs) such as ResNet50 and VGG16 have achieved superior accuracy compared to traditional machine learning methods. This study explores feature extraction using pre-trained models, the implementation of classifiers like Support Vector Machines (SVM) and K-Nearest Neighbors (KNN), and the integration of multi-view learning techniques, such as early fusion. The results demonstrate the effectiveness of DL and TL in improving classification performance, highlighting their significant potential to advance global healthcare diagnostics.

1. INTRODUCTION

The rapid advancements in machine learning (ML) and deep learning (DL) have significantly transformed artificial intelligence (AI), particularly in the field of image classification. DL models, particularly Convolutional Neural Networks (CNNs), have demonstrated remarkable accuracy, often surpassing human capabilities in complex classification tasks. This progress has been driven by several key factors, including the increasing availability of large labelled datasets, improvements in GPU-enabled high-performance computing, and the continuous evolution of neural network architectures. These advancements have enabled the efficient training of deep networks, significantly reduced computation time while enhancing predictive accuracy.

In recent years, DL-based image classification has gained prominence in various domains, particularly in medical diagnostics, autonomous systems, and security applications. The ability of DL models to automatically extract hierarchical features from images allows them to outperform traditional machine learning approaches, especially when large and well-annotated datasets are available [1]. Additionally, the emergence of Transfer Learning (TL) has further revolutionized the field by enabling the reuse of pre-trained models for new tasks, reducing the need for extensive labelled data and computational resources.

With DL evolving at a rapid pace, its applications in image analysis continue to expand, shaping the future of AI-driven data interpretation. In this chapter, we explore the fundamental principles of DL and TL, their significance in image classification, and their implications for this study, particularly in the context of medical image analysis.

2. OVERVIEW OF DEEP LEARNING

DL is a subfield of machine learning that utilizes artificial neural networks with multiple layers to model and solve complex problems. Its primary goal is to enable computers to learn from experience by processing vast amounts of data and making predictions or decisions based on that learning. Unlike traditional machine learning methods, deep learning networks

*Corresponding author email: t.sakirin87@gmail.com
DOI: https://doi.org/10.70470/EDRAAK/2025/006

automatically extract features from raw data such as images, audio, or text without requiring manual feature engineering. This is achieved through backpropagation, a process that adjusts the network's weights to minimize the error between its predictions and actual outputs.

DL has led to groundbreaking advancements in various fields, including image recognition, speech recognition, natural language processing, and recommendation systems. The performance of deep learning models improves as they are exposed to more data, much like how humans learn by analyzing examples and refining their understanding through experience [2]. Since deep learning algorithms learn features and tasks directly from the training data, they eliminate the need for domain experts to manually extract relevant features [3].

In medical image classification, deep learning models rely on labeled datasets, where each image is tagged as COVID-19, viral pneumonia, or normal (healthy lungs). The labeled images enable the model to learn specific patterns and features, allowing it to classify new input images accurately. Deep learning models are built using neural network architectures, which is why they are often referred to as deep neural networks (DNNs). The term "deep" refers to the number of hidden layers in an artificial neural network (ANN). One of the most widely used DNN architectures for image data analysis is the Convolutional Neural Network (CNN) [4], which excels in feature extraction and classification tasks.

3. OVERVIEW OF TRANSFER LEARNING

Instead of training a model from scratch for a new task, transfer learning allows us to leverage the knowledge acquired by a pre-trained model on a similar task and fine-tune it for the new application. This approach has been successfully applied across various domains, including computer vision, natural language processing, and speech recognition [5].

Transfer learning significantly reduces the amount of training time and data required to achieve high performance. By utilizing learned features from a well-trained model, it accelerates convergence and enhances model efficiency. Additionally, transfer learning improves generalization and helps mitigate overfitting where a model becomes too specialized in the training data and struggles to perform well on unseen data.

Deep learning models typically require large amounts of labeled data to achieve optimal performance. However, in many real-world scenarios, obtaining sufficient training data is challenging. Transfer learning addresses this issue by enabling models to adapt to new tasks with limited data while maintaining high accuracy [3]. This makes it particularly valuable in domains where data collection is expensive or time-consuming, such as medical imaging and rare disease classification.

4. FEATURE EXTRACTION

Feature extraction is the process of extracting relevant information from raw data to improve the performance of machine learning models. It is a crucial step in machine learning, as it involves transforming raw data into a set of meaningful features that serve as inputs for model training. The primary goal of feature extraction is to reduce the dimensionality of the dataset while preserving the most important characteristics, thereby improving computational efficiency and model accuracy.

Effective feature extraction simplifies data representation by eliminating irrelevant or redundant features, which helps prevent overfitting and enhances the model's ability to generalize to new data. The choice of feature extraction techniques depends on the type of data and the specific problem being addressed. Traditional methods include statistical approaches and wavelet transforms, while deep learning techniques, such as Convolutional Neural Networks (CNNs), have revolutionized feature extraction, particularly for image and audio processing. By automatically learning hierarchical representations, CNNs have significantly improved the accuracy and efficiency of tasks such as image recognition, speech analysis, and natural language processing.

4.1 PRE-TRAINED MODEL:

A pre-trained model is a machine learning model that has been trained on a large dataset and saved for reuse in various tasks, eliminating the need for retraining from scratch. These models serve as a strong foundation for training new models or making predictions on new data, significantly reducing computational costs and training time.

In image classification tasks, pre-trained models are often used for feature extraction by removing the output layer responsible for classification. The remaining layers function as a static feature extractor for new datasets, capturing complex patterns and hierarchical features from images. Convolutional Neural Networks (CNNs) are among the most widely used pre-trained models, demonstrating state-of-the-art performance across numerous benchmark datasets. Prominent architectures such as VGGNet, GoogLeNet (Inception), and ResNet have proven highly effective in extracting deep representations from image data due to their deep convolutional layers followed by fully connected layers. These architectures have been extensively applied in medical imaging, achieving remarkable results in detecting diseases such as cancer, diagnosing diabetic retinopathy from retinal scans, and identifying lung diseases from chest X-rays [6].

To further enhance the accuracy of medical image classification, pre-trained models are often combined with advanced techniques such as data augmentation, clustering, and attention mechanisms. Data augmentation techniques; including rotation, scaling, and flipping, help increase the diversity of training data, improving model generalization and flexibility.

Additionally, ensemble learning, which integrates predictions from multiple models, enhances classification accuracy and reliability. Attention mechanisms further refine the model's performance by allowing it to focus on specific regions of interest within medical images, improving the precision of disease localization and classification [7].

4.2 ResNet50 Model:

ResNet50 is a pre-trained CNN model designed by Microsoft Research for image classification using DL techniques. It belongs to the ResNet (residual network) family, which addresses the issue of vanishing gradients in deep neural networks. The model was trained using the ImageNet dataset consisting of 50 layers and contains more than a million labeled images and more than 1,000 distinct classes and it was training their algorithm taught to accurately predict the appropriate classification for the given image.
The ResNet50 model can assist as an initial reference for various computer vision tasks such as object identification, image segmentation, and image recognition. We can achieve a high level of accuracy with a very small amount of training data by optimizing a model with a selected dataset, and in many deep learning frameworks, we can have found them, such as TensorFlow and PyTorch. These frameworks can load the ResNet50 pre-trained model, which can then be fine-tuned to fit specific tasks. Figure 1 shows the block diagram structure of the ResNet50 model.

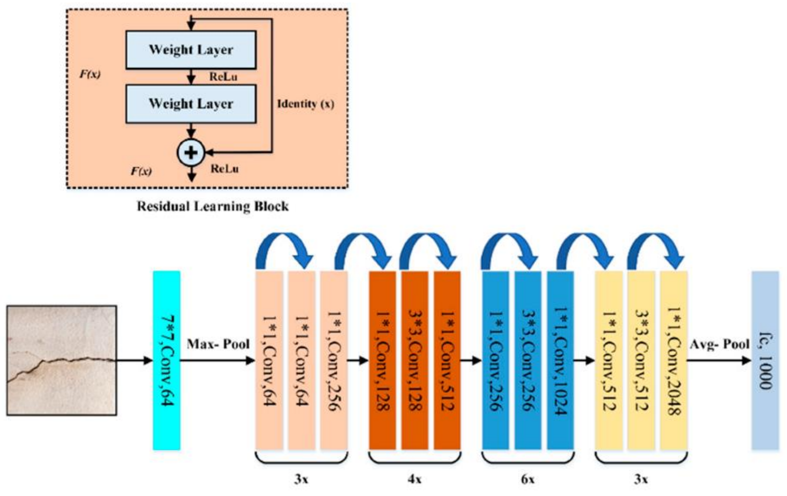

Fig 1: Architecture of ResNet50 Pre-trained CNN [8]

These steps provide a detailed explanation of the process for utilizing the ResNet50 pre-trained model to extract picture characteristics.

• Step 1: Ensure that suitable libraries, such as TensorFlow/Keras, are inserted to use ResNet50 and process the image.

```
from tensorflow.keras.applications.resnet import ResNet50
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
```

• Step 2: Load the pre-trained ResNet50 model:

```
resnet = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

We begin with the model with pre-trained weights extracted from 'imagenet'. The `include_top=False` means that we are excepting the top classification layers (fully connected) of ResNet50. By putting `include_top` to False, we retain the convolutional base of the model, which permits us to utilize it for feature extraction rather than classification.

• Step 3: Stop the layers of ResNet50:

```
for layers in resnet.layers:
    layers.trainable=False
```

In the loop
`for layers in resnet.layers: layers.trainable=False`
each layer of the ResNet50 model is repeated, and its `trainable` feature is set to False. This step freezes the weights of all layers in ResNet50 so that they won't be up to dated during the training process.

• Step 4: Preprocess and extract features from images:

```
for path in train_df['filename'].to_numpy():
```

Processes the images specified in the ‘train_df’ DataFrame. For each image file path in the ‘filename’ column of the DataFrame, it performs the following steps:

1. Load the image and resize it:

```
input_shape=(224, 224, 3))
```

The image is loaded using the `load_img` function from Keras and resized to the target size of `(img_size, img_size)`, which is `(224, 224)` in this case, matching the input shape of ResNet50.

2. Convert the image to an array and expand its size:

```
img_array = img_to_array(x)
img_array = np.expand_dims(img_array, axis=0)
```

The image is transitioned to a NumPy matrix using `img_to_array` function from Keras. The matrix is then extended along the first dimension (axis=0) to be fitted with ResNet50’s input shape.

3. Extract features using ResNet50:

```
features_res = resnet.predict(img_array)
```

We input the preprocessed image array into the ResNet50 model using the `resnet.predict(img_array)` method. This stage uses the model to extract the features from the image.

• Step 5Append the extracted features to ‘feature_list2’:

```
feature_list2.append(features_res)
```

In the variable ‘features_res’ we stored the features extracted from each image. Those features are added to the list ‘feature_list2’, making a list of characteristic vectors for all the images.

Step 6: Reshape the feature list:

```
feat_lst2 = np.reshape(feature_list2,(-1,7*7*2048))
```

The list ‘feature_list2’ is organized into a NumPy array ‘feat_lst2’ with dimensions (-1, 7 * 7 * 2048) to transform the list of feature vectors into a two-dimensional array, where each row fits the feature vector of one image.
The variable ‘feat_lst2’ has the image features that are retrieved using the ResNet50 model. These properties may be used for a range of activities, including classifying images without the requirement to do new analysis of them on every round.

4.3 VGG16 Model:

VGG16 is a pre-trained CNN model by the Visual Geometry Group (VGG) at the University of Oxford. The structure is a 16-layer deep learning model, consisting of 13 convolutional layers and 3 fully connected layers.
The VGG16 model underwent training using the ImageNet dataset, which consists of more than one million pictures that have been labeled. The model has demonstrated exceptional precision in many image recognition tasks, encompassing object detection and picture categorization. The VGG16 Pre-trained CNN architecture, seen in figure (2), is distinguished by its use of compact 3x3 filters that are iteratively applied to the input picture for the purpose of extracting features at varying levels of abstraction. The model also employs max pooling layers to decrease the resolution of the feature maps and diminish the complexity of the input.
By using the ImageNet dataset, they trained the VGG16 model, which consists of more than a million labeled images. It has shown special accuracy in several image recognition tasks, including object detection and image classification. The VGG16 pre-trained CNN architecture, shown in Figure 2, features built-in 3x3 filters that are frequently applied to the input image to extract features at different levels of removal. The model also uses maximum pooling layers to reduce the accuracy of the feature maps and simplify the input.
By taking advantage of the pre-trained weights of the model, it is possible to achieve high performance on image recognition tasks, even when the amount of training data is low. The VGG16 model is used in computer vision applications, and many deep learning frameworks provide pre-trained repetition of the model that can be used for transfer learning. A max-pooling layer follows each group of 13 convolutional layers. The VGG-16 design takes an RGB image with dimensions of 224 x 224 pixels as its input. After processing the input through a series of convolutional layers and max pooling, a dimensional feature map (7,7,512) is generated and then converted into a flat feature vector of dimensions (1-25088). The flattened feature is then fed to three perfectly connected layers with identical configurations, resulting in a feature vector of size (14096). The output of the fully connected layers is then used as input to the SoftMax layer, which is responsible for classification.

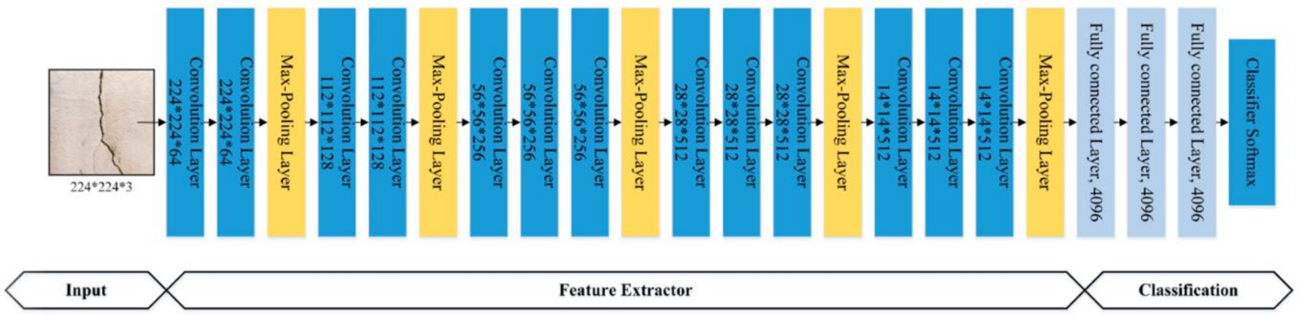
224*224*3
Convolution Layer 224*224*64
Convolution Layer 224*224*64
Max-Pooling Layer
Convolution Layer 112*112*128
Convolution Layer 112*112*128
Max-Pooling Layer
Convolution Layer 56*56*256
Convolution Layer 56*56*256
Convolution Layer 56*56*256
Max-Pooling Layer
Convolution Layer 28*28*512
Convolution Layer 28*28*512
Convolution Layer 28*28*512
Max-Pooling Layer
Convolution Layer 14*14*512
Convolution Layer 14*14*512
Convolution Layer 14*14*512
Max-Pooling Layer
Fully connected Layer, 4096
Fully connected Layer, 4096
Fully connected Layer, 4096
Classifier Softmax
Input
Feature Extractor
Classification

Fig 2: Architecture of VGG16 Pre-trained CNN [9]

The following are the sequential instructions for using the VGG16 pre-trained model for extracting picture features:

1. Import the VGG16 model and any other required libraries.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
```

2. Load the pre-trained VGG16 model with weights trained on the ImageNet dataset and exclude the top fully connected layers:

```
vgg = VGG16(weights='imagenet', include_top=False)
```

Here, the weights=’imagenet’ argument means the pre-trained weights on the ImageNet dataset will be loaded and include_top=False means the top dense layers of the VGG16 model, which are typically used for classification, will not be included. Set all layers in the VGG16 model to non-trainable (frozen) to prevent them from being updated during training:

```
for layer in vgg.layers:
    layer.trainable = False
```

This loop iterates over all layers in the VGG16 model and sets their trainable property to False. Print the output layer of the VGG16 model:

```
print(vgg.output)
```

This will print the output tensor of the VGG16 model after removing the top layers, initialize an empty list feature_list to store the extracted features and iterate through each image path in the train_df[‘filename’] column (assuming train_df is a Data Frame containing file paths to the images):

```
for path in train_df['filename'].to_numpy():
```

3. Load and preprocess the image using load_img and img_to_array functions:

```
x = load_img(path, target_size=(img_size, img_size))
    img_array = img_to_array(x)
```

The image is loaded and converted to a NumPy array. Now expand the dimensions of the image array to match the input shape required by the VGG16 model (4D array with batch size 1):

```
img_array = np.expand_dims(img_array, axis=0)
```

4. Use the VGG16 model to predict the features of the input image:

```
features = vgg.predict(img_array)
```

The predict method of the VGG16 model will return the extracted features for the given image.

5. Append the extracted features to the feature_list:

```
feature_list.append(features)
```

6. After processing all images, reshape the feature_list to a 2D array:

```
feat_lst = np.reshape(feature_list, (-1, 7 * 7 * 512))
```

Here, -1 in the reshaping indicates that the size of that dimension will be automatically inferred based on the other dimensions. The resulting feat_lst will have shape (number_of_images, 7 * 7 * 512), where each row represents the flattened feature vector for a single image.

In summary, these steps use the pre-trained VGG16 model to extract image features from a dataset of images and store the features in a 2D NumPy array fatalist. These features can be used as input to other machine learning models or for various computer vision tasks like image classification.

5. SUPPORT VECTOR MACHINE (SVM) CLASSIFIER

SVM is a powerful and popular algorithm used for classification and regression analysis. In SVM classification, the goal is to divide a set of data points into classes by finding a hyperplane that separates the data points in the most optimal possible way. The optimal hyperplane is the one that maximizes the margin between the two classes.

The margin is the distance between the hyperplane and the nearest data point from each class. The SVM algorithm aims to find the hyperplane that maximizes this margin, as it provides the best generalization performance. In other words, the SVM algorithm aims to find the hyperplane that is most robust to future data points.

SVM classification works by mapping the input data into a higher-dimensional feature space where a linear separation is possible. This mapping is achieved using a kernel function, which transforms the input data into a higher-dimensional space. Once the data is mapped, the SVM algorithm finds the hyperplane that separates the data points in the most optimal way possible.

SVM classification has been successfully applied in many real-world applications such as text classification, image classification, and bioinformatics. It is a powerful and flexible algorithm that can handle both linearly separable and non-linearly separable data, making it a popular choice in machine learning. These steps demonstrate how to train an SVM (Support Vector Machine) classifier using features extracted from a pre-trained model (ResNet50 or VGG16) for the classification of COVID-19, Normal, and Viral Pneumonia images:

1. Train-Test Split:

```
X_train#, X_test#, y_train#, y_test# = train_test_split(feat_#, y, test_size=0.1, random_state=2020)
```

This line uses the `train_test_split` function from `sklearn.model_selection` to split the dataset into training and testing sets. `feat_lst` contains the features extracted from the images using the pre-trained model, and `y` represents the corresponding labels (COVID, Normal, or Viral Pneumonia). The `test_size =0.1` parameter indicates that 10% of the data will be used for testing, and the remaining 90% will be used for training and validating. The `random_state=2020` sets a random seed to ensure reproducibility.

2. Creating the SVM model:

```
    glm2 = svm.SVC(kernel='linear', random_state=2020)
```

In this line, an SVM classifier is created with the `svm.SVC` class from `sklearn.svm`. The `kernel='linear'` parameter specifies that a linear kernel will be used for the SVM, which means the decision boundary will be a linear hyperplane. The `random_state=2020` sets the random seed for reproducibility of the model's results.

3. Training the SVM model:

```
#_svm = glm2.fit(X_train#, y_train#)
```

The `fit` method is called to train the SVM model on the training data (`X_train#` and `y_train#`). The model will learn to classify the images into the appropriate classes based on the features extracted from the pre-trained model.

4. Evaluating the SVM model:

```
    #_svm_score = glm2.score(X_test#, y_test#)
```

The `score` method is used to evaluate the trained SVM model on the testing data (`X_test#` and `y_test#`). It calculates the accuracy of the model on the test set. The `#_svm_score` variable will hold the accuracy score.

At this point, the SVM model is trained using the features extracted from the pre-trained model (ResNet50 or VGG16), and its performance is evaluated on the test data to determine how well it can classify the images into COVID-19, Normal, and Viral Pneumonia categories based on the features it has learned during training.

6. K-NEAREST NEIGHBORS (KNN) CLASSIFIER

KNN is a type of classification algorithm used in machine learning and data mining. It is a non-parametric algorithm, meaning that it does not make any assumptions about the underlying data distribution.
The KNN algorithm works by finding the K-nearest data points in the training set to a given test data point, based on a distance metric such as Euclidean distance or cosine similarity. It then assigns the test data point to the class that is most common among its K-nearest neighbors. For example, if K=5 and the five nearest neighbors to a test data point are all members of Class A, then the algorithm would predict that the test data point also belongs to Class A. KNN is a simple and effective algorithm that can be used for both binary and multi-class classification problems. However, its performance can be sensitive to the choice of K and the distance metric used. Additionally, KNN can be computationally expensive, particularly with large datasets.
These steps demonstrate how to use the K-Nearest Neighbors (KNN) classifier in Python for the classification of COVID-19, Normal, and Viral Pneumonia images based on the features in `X_train` and their corresponding labels in `y_train`. Let's break down the steps:

1. Importing the necessary module:

```
from sklearn.neighbors import KNeighborsClassifier
```

Before running this code, you need to import the `KneighborsClassifier` class from the `sklearn.neighbors` module to use the KNN classifier.

2. Creating the KNN model and training it:

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

Here, a KNN classifier object `knn` is created using the `KneighborsClassifier()` constructor. The default value for `n_neighbors` (the number of neighbors to consider) is 5, which means it will consider 5 nearest neighbors for classification. We can specify the value of `n_neighbors` by passing it as an argument when creating the KNN classifier, like `KneighborsClassifier(n_neighbors=3) `.
The `fit` method is then called to train the KNN classifier on the training data `X_train` and their corresponding labels `y_train`. The classifier learns the relationships between the features and their corresponding classes during this training process.

3. Printing the accuracy of the KNN model:

```
_knn_Score = knn.score(X_test, y_test)
print("Accuracy obtained by K Neighbors Classifier model:", _knn_Score)
```

The `score` method is used to calculate the accuracy of the trained KNN model on the test data `X_test` and `y_test`. It compares the predicted labels with the true labels and calculates the accuracy as the ratio of correct predictions to the total number of samples in the test set. The accuracy is then printed to the console.

4. Making predictions using the KNN model

```
ypred_knn = knn.predict(X_test)
```

The `predict` method is used to make predictions on the test data `X_test`. It returns an array `ypred_knn` containing the predicted labels for the test samples based on the trained KNN model.
At this point, the KNN classifier has been trained on the training data, and its accuracy has been evaluated on the test data. The `ypred_knn` array holds the predicted labels for the test samples, which can be further used for evaluation or analysis purposes.

7. MULTI-VIEW LEARNING

Multi-view learning is a type of machine learning where multiple views or representations of the same data are used to improve the accuracy of the learning algorithm. The idea behind multi-view learning is that different views of the same data can provide complementary information, which can be used to better capture the underlying structure of the data. Multi-view learning can be applied in a variety of settings, including image and video processing, natural language processing, and bioinformatics. In image processing, for example, multiple views of an image might include its pixel values,

color histogram, and texture features. In natural language processing, multiple views of a document might include its bag-of-words representation, its part-of-speech tags, and its syntactic parse tree.
In general, multi-view learning is a powerful technique for improving the accuracy of machine learning algorithms, particularly in settings where multiple sources of information are available.

7.1 Early Fusion model

Early fusion is a type of features concatenation technique used in machine learning, particularly in computer vision tasks such as image classification or object recognition to deal with multi-view data. In early fusion, the features of the input data are combined at the earliest possible stage of the model. For example, in image classification, the input image may be represented as a matrix of pixel values. In early fusion as in figure 3 below, these pixel values are combined with other relevant information such as color histograms, texture features, or edge information, and fed into the model as a single, unified input. This allows the model to learn more complex representations of the input data and make better predictions.

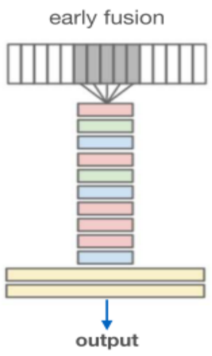

Fig 1: The Architecture of Early Fusion [10]

8. CONCLUSION

This chapter provided a comprehensive overview of deep learning (DL) and transfer learning (TL) techniques, emphasizing their pivotal roles in the classification of medical images. By leveraging pre-trained models such as ResNet50 and VGG16, the study demonstrated how feature extraction can be effectively performed, facilitating accurate classification of diseases like COVID-19 and viral pneumonia. The integration of classifiers like Support Vector Machines (SVM) and K-Nearest Neighbors (KNN) further enhanced the model's performance, showcasing the synergy between deep learning architectures and traditional machine learning algorithms.
Moreover, the exploration of multi-view learning, particularly the early fusion model, highlighted the benefits of combining multiple feature representations to capture the intricate details of medical images. This approach not only improves the accuracy of classification but also provides a more holistic understanding of the data, which is crucial in medical diagnostics.
The advancements in GPU-enabled computing and the availability of large labeled datasets have significantly contributed to the success of deep learning models, making them indispensable tools in modern AI applications. As DL and TL continue to evolve, their applications in medical image analysis are expected to become even more sophisticated, offering enhanced diagnostic capabilities and contributing to better healthcare outcomes globally.
Future research should focus on optimizing these models for real-time applications, addressing challenges related to data privacy and security, and exploring the potential of emerging techniques like attention mechanisms and ensemble learning to further bolster the accuracy and reliability of medical image classification systems.

Funding:

The study was conducted independently without any monetary support from third-party organizations or sponsors. The authors declare that no external funding was provided for this research.

Conflicts of Interest:

The authors declare no competing interests.

Acknowledgment:

The authors would like to thank their institutions for their moral support and provision of critical resources throughout the research process.

References

[1] Hosseinzadeh. H, "Deep multi-view feature learning for detecting COVID-19 based on chest X-ray images," *Biomed. Signal Process. Control*, vol. 75, p.p 103595, 2022.

[2] Mishra. M., Choudhury. T., and Sarkar. T, "CNN based efficient image classification system for smartphone device," *Electronic Letters on Computer Vision and Image Analysis,* vol. 0(0), pp.1-7, 2021.

[3] Alzubaidi, L., Zhang, J., Humaidi, A.J. et al. Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *J Big Data 8*, vol,53 (2021), DOI: 10.1186/s40537-021-00444-8S.

[4] Kugunavar, S. and Prabhakar, C.J., Convolutional neural networks for the diagnosis and prognosis of the coronavirus disease pandemic. *Vis. Comput. Ind. Biomed*. Vo. 4 (12), 2021, DOI:10.1186/s42492-021-00078-w

[5] Chaddad. A., Hassan. L., and Desrosiers. C, "Deep CNN models for predicting COVID-19 in CT and X-ray images," *Journal of Medical Imaging,* Vol. 8(S1), 2021, DOI: 10.1117/1.JMI.8.S1.014502

[6] S. Roy *et al.*, "Deep learning for classification and localization of COVID-19 markers in point-of-care lung ultrasound," *IEEE Trans. Med. Imag.*, vol. 39, no. 8, pp. 2676–2687, 2020, DOI: 10.1109/TMI.2020.2994459.

[7] Abbas. M., Abdelsamea. M, and Gaber. M.M, "Classification of COVID-19 in chest X-ray images using DeTraC deep convolutional neural network," *Appl. Intell.*, vol. 51, pp. 854–864, 2021.

[8] Ali. L., Alnajjar. F., Jassmi. H.A., Gocho. M., Khan. W. and Serhani. M.A, Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures, *Sensors,* vol. 21(5), 2021, DOI: 10.3390/s21051688

[9] Snoek. G., Worring. M, and Smeulders. A. W, "Early versus late fusion in semantic video analysis," in *Proc. 13th Annu. ACM Int. Conf. Multimedia* , pp. 399–402, 2005.

[10] Mohammad. R and Abolfazl. A, "A new modified deep convolutional neural network for detecting COVID-19 from X-ray images," *arXiv preprint arXiv:2004.08052*, 2020.