Classification of Chest X-Rays Using Shallow and Deep Learning Methods

Kopal Garg
Department of Computer Science
University of Toronto
kopal.garg@mail.utoronto.ca

Rohan Deepak Ajwani
Department of Electrical & Computer Engineering
University of Toronto
rohan.ajwani@mail.utoronto.ca

Yash Prakash
Department of Computer Science
University of Toronto
yash.prakash@mail.utoronto.ca

Charita Koya
Department of Computer Science
University of Toronto
charita.koya@mail.utoronto.ca

I. INTRODUCTION

As a result of a deadly coronavirus that took the form of a pandemic, hundreds of thousands of deaths have been recorded due to improper and untimely diagnosis of COVID-19 in patients. During this time, all of the world's focus shifted towards providing the best healthcare for anyone that is affected. Our study is based on one such major goal—to distinguish and classify between chest X-Ray images of normal, pneumonia-ridden, and COVID-19 affected lungs. In this work, we show how effective shallow and deep learning methods can be in assisting in the diagnosis of chest X-Rays of diseased lungs into the aforementioned three categories. We worked with a balanced dataset consisting of 6939 .jpeg image samples of chest X-Ray posteroanterior images. In our first task, we trained shallow classification models such as Support Vector Machine (SVM), XGBoost and Decision Tree (DT) and compared them against various CNN configurations, while experimenting with various hyperparameters, loss functions and gradient descent optimization algorithms. In our second task, we demonstrated the use of shallow classifiers in classifying features extracted by a CNN. We removed the fully-connected layer of the CNN, while maintaining the remaining network that consisted of a series of convolution and pooling layers, and added classifiers like SVM and XGBoost on top. For both tasks, we employed standard image processing techniques such as histogram equalization, grayscale transformation and fixed thresholding and also experimented with dimensionality reduction techniques like Principal Component Analysis (PCA).

Keywords: *Chest X-Ray Classification, X-Ray Image Pre-processing, Convolutional Neural Networks, Transfer Learning*

II. RELATED WORK

Several studies have reported the use of machine and deep learning approaches for investigation of COVID-19 chest X-Rays. Using a dataset of 225 confirmed COVID-19 X-Ray scans, Sekeroglu et al. [1] performed 38 experiments using 4 ConvNet architectures differing in input dimension, and dense layer number and dimension, 10 experiments using five ML models such as Support Vector Machine (SVM), Logistic Regression (LR), Naïve Bayes (NB), Decision Trees (DT) and k-Nearest Neighbor, and 14 transfer learning experiments using pre-trained networks. They achieved a mean sensitivity of 93.84%, mean specificity of 99.18%, and mean accuracy of 98.50%. Using a dataset of 1341 normal, 1345 viral pneumonia and 219 COVID-19 patients' chest X-Ray images, Satu et al. [2] proposed the use of an enhanced convolutional neural network (CNN) that achieved a 94.03% accuracy, 95.52% AUC and 94.03% F1-measure in detecting COVID-19 from chest X-Ray images. However, high accuracy on a heavily imbalanced and small dataset cannot guarantee the effectiveness of COVID-19 detection. Ozturk et al. [3] proposed a variant of a similar CNN architecture that achieved 98.08% test accuracy in binary classification and 87.02% test accuracy in three-category classification (COVID-19, pneumonia and normal cases).

Minaee et al. [4] trained four popular CNN models, ResNet18, ResNet50, SqueezeNet, and DenseNet-121, on a dataset containing 5000 chest X-ray images with binary labels. Both ResNet18 [5] and ResNet50 [6] were pre-trained on the ImageNet dataset and differed in their number of layers. SqueezeNet [7] employs model compression techniques that alternate between having 1x1 layers to squeeze incoming data, and two parallel 1x1 and 3x3 layers to expand the data. In DenseNet [8], each layer receives feature maps from preceding layers with fewer channels. DenseNet is also known to have higher computational and memory efficiency. In this study's dataset, since the number of scans in the COVID-19 class were limited, the last layer of their CNN was fine-tuned, and the pre-trained models were used as a feature extractor. Their best model achieved a sensitivity of 98% while having a specificity of 92%.

Sethy et al. [9] also performed transfer learning using pre-trained networks including AlexNet, VGG16, VGG19, GoogleNet, ResNet101, InceptionV3, MobileNetV2 and ShuffleNet. They extracted features from the fully connected layers of pre-trained networks and fed them to an SVM classifier for multi-class classification. Of interest, they achieved an accuracy of 98.66% using ResNet50 and SVM. Stubblefield et al. [10] used a deep CNN, CheXNet, for extracting image features and XGBoost for performing classification. By using the output vector from CheXNet as input for the XGBoost model, they transferred high-level latent representations of the chest X-Ray image's features. This decreased the amount of time needed to train the XGBoost model. Rajagopal et al. [11] performed a similar comparative study on four different models: transfer learning model (VGGNet), CNN, SVM using features extracted from a CNN, and XGBoost with features extracted from a CNN, achieving an accuracy of 95.27%, 95.52%, 94.94% and 95.71%, respectively. Although deep learning techniques enable image classification without manual feature engineering, Khuazni et al. [12] showed that by employing dimensionality reduction techniques, such as kernel-PCA, one can generate a set of optimal features and considerably decrease their model's training time due to lower redundancy. They also showed that their model had ~10,000 parameters, which is considerably smaller than typical classification models like AlexNet with 60M parameters, and ResNet50 with 25M parameters.

III. PROPOSED APPROACH

A. Data Pre-Processing

The data was collected from the following sources: firstly, a total 1401 samples of COVID-19 were collected using GitHub repository [13], [14] and Figshare data repository websites [15], [16]. 912 augmented images were also collected from Mendeley [17]. Then, 2313 samples of normal and pneumonia XRays were obtained from Kaggle [18], [19].

Since we acquired images from various sources, we applied a few preprocessing steps before training to prepare images for classification. The dataset consists of images with variable-resolutions, color channels, and sizes. To standardize these characteristics across the dataset, we converted the images to single channel grayscale, and resized them to 128 x 128.

To handle images with varying illumination, we used a function from OpenCV that applies fixed-level thresholding to a single-channel array. It outputs a binary image out of a grayscale image and removes noise by filtering out pixels with extreme values. A fixed threshold is computed using the maximum and minimum pixel value in an image. If a given pixel value is smaller than the threshold, it is set to 0, otherwise, it is set to the maximum value of 255. This is also useful in segmenting the region of interest, i.e., the lungs, from the background.

We noticed that images from certain datasets had brighter regions, i.e., pixel values were confined to a high range of values, while others were darker. We used histogram equalization from OpenCV to improve the contrast of input grayscale images, and to stretch out the intensity range. This is a transformation function that maps pixels in brighter regions to pixels in full regions, and outputs an image with consistent lighting conditions. Figure 1 shows a comparison between an original image and an image obtained using our pre-processing pipeline.

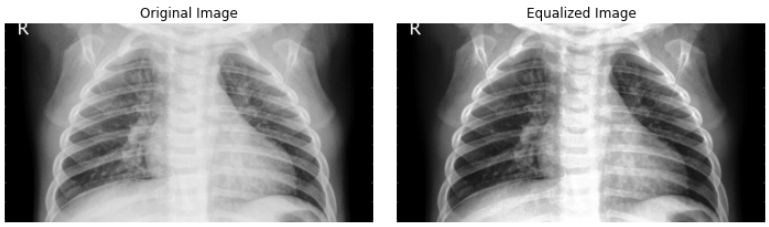

FIGURE 1: A. Original image. B. Image after grayscale transformation, resizing, fixed thresholding, and histogram equalization.

Finally, we experimented with PCA as a feature space reduction technique. Although our dataset is balanced, and we use more images than most existing studies, operating in a 16,384-dimensional space can be problematic with a small sample size. Therefore, we used PCA to reduce the dimensionality of the feature space by calculating the eigenvectors of the covariance matrix of the set of 16,384-dimensional feature vectors, and then projecting each feature vector onto the largest eigenvectors. The scikit-learn.decomposition.PCA documentation states that the number of PCs must be between 0 and min(# training features, # training samples). Our training set had 1,892 samples and 16,384 features. Therefore, the maximum number of PCs we could use was 1,892. We empirically determined an optimal number of PCs, by finding the number that led to high classification accuracy, and minimum total reconstruction loss over all m training points, which was computed using the Euclidean distance:

$R = \sum_{i=1}^{m} \left|\left| x_i - \hat{x}_i \right|\right|^2$. By visualizing the plots in Figure 2., we determined that 32^2 would be an optimal number of PCs. By employing PCA, we converted the original pool of 16,384 (i.e., 128 x 128 image) features to 1,024 (i.e., 32 x 32 image) synthetic features resulting in a ~10x smaller feature space, while still capturing >99% of the overall variance. Figure 3 shows a comparison between a preprocessed image and the inverse transformed image after PCA.

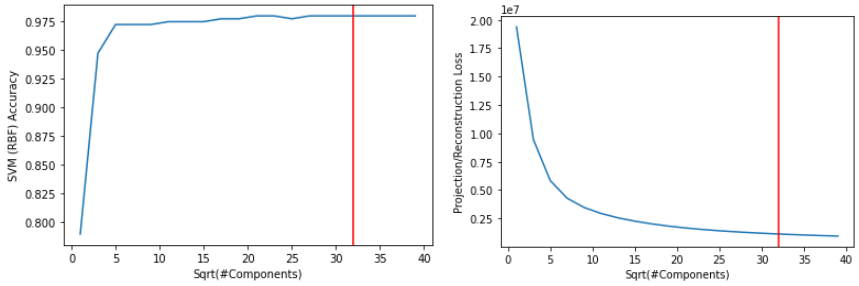

FIGURE 2: A. Baseline model accuracy. B. Projection loss.

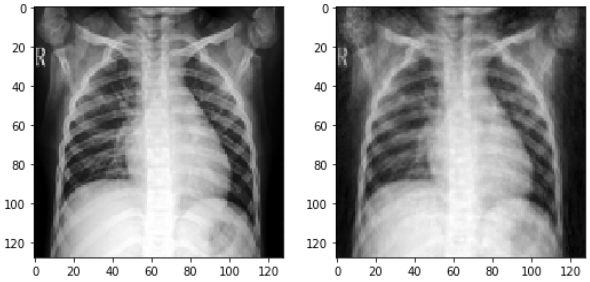

FIGURE 3: A. Pre-processed image before PCA. B. Reconstructed image after inverse PCA.

B. Baseline Models

1) Support Vector Machines

SVMs are inherently two-class classifiers but can be extended to handle multiclass classification through the one-vs-rest (OVR) method, which involves training a series of k binary SVMs, k being the number of classes. The mth classifier is trained with all of the examples in the mth class with positive labels, and the rest of the examples with negative labels. For a given training set with l data points, $(x_1, y_1),\ldots,(x_l, y_l)$, where $x_i \in R^n$, $i = 1,..,l$ and $y_i \in \{1,\ldots,k\}$, the mth classifier solves the following optimization problem: $min_{w^m,b^m,\xi^m} \frac{1}{2}(w^m)^T w^m + C\sum_{i=1}^{l} \xi_i^m$ with the following constraints: $(w^m)^T \phi(x_i) + b^m \geq 1 - \xi_i^m$, $if\ y_i = m$ and $(w^m)^T \phi(x_i) + b^m \leq 1 - \xi_i^m$, $if\ y_i \neq m$, and $\xi_i^m \geq 0$, $i = 1,..,l$, where ϕ map x_i to a higher dimension, C is the penalty parameter and ξ are slack variables that make it possible to satisfy the constraints even when the original constraint is not met [20]. Smaller C values emphasize the importance of ξ and larger C diminish the importance of ξ. By minimizing this, we maximize the margin between two classes. This leads to k decision functions and the test datum is accepted by the class with the highest value of decision function and rejected by the rest, such that the $class_x = argmax_{m=1,\ldots,k}((w^m)^T \phi(x) + b^m)$ [20]. We performed cross-validation grid-search using the $GridSearchCV$ module from *scikit-learn*, considering three hyperparameters: C [0.01, 0.1, 1, 10, 100], $kernel$ ['rbf', 'poly', 'linear', 'sigmoid']and $gamma$ [1, 0.1, 0.01, 0.001, 0.0001]for non-linear kernels. It was found that {'C': 0.01, 'gamma': 1, 'kernel': rbf} were the optimal hyperparameter values.

2) *Decision Tree*

Decision Trees (DTs) aim to predict target values by learning rules inferred from the features, and can inherently handle multiclass classification problems. Without assuming features are uncorrelated, the method builds a single model that can simultaneously predict all outputs. The method works well with balanced datasets like ours, as it prevents the tree from being biased towards a single dominant class. Again for a given training set with l data points, (x_1, y_1),..., (x_l, y_l), where $x_i \in R^n$, $i = 1,..,l$ and $y_i \in \{1,...,k\}$, a DT would recursively partition the feature space resulting in samples with similar target values being grouped. For data at node m, (Q_m), the data would be partitioned into $Q_m^{left}(\theta)$ and $Q_m^{right}(\theta)$ subsets, where $Q_m^{left}(\theta) = \{(x, y)|x_j \leq t_m\}$ and $Q_m^{right}(\theta) = Q_m \backslash Q_m^{left}(\theta)$ where j represents a feature and t_m represents a threshold [21]. The quality of the split would be determined using functions that aim to find parameters that minimize impurity [21]. We performed a 10-fold cross-validation grid search on impurity $criterion$ with values ['gini','entropy']and $max\ depth$ with values ranging from 1 to 50, at a step size of 5. It was found that {'criterion':'gini', 'max_depth': 30}were the optimal hyperparameter values.

3) *XGBoost*

eXtreme Gradient Boosting (XGBoost) is based on the gradient boosting decision tree algorithm but employs an enhanced regularization method. It builds simple weak learning decision trees iteratively and improves upon them to create a strong classifier. By default, *XGBClassifier* implements OVR classification. We performed a grid search using 10-fold cross-validation, on the $n_{estimators}$ argument, evaluating a series of values from 50 to 200 at a step size of 50 and the $max\ depth$ parameters ranging from 1 to 11, at increments of 2. It was found that {'max_depth': 3, 'xgb__n_estimators': 50}were the optimal hyperparameter values.

C. Baseline CNN Architecture

Convolutional Neural Networks (CNN) can use backpropagation to adaptively learn spatial hierarchies of features [22]. In our base implementation, we incorporated 2 convolutional layers, a pooling layer and two fully-connected layers along with regulatory units like dropout and operations like flattening to optimize CNN performance.

A convolutional layer works by dividing the image into receptive fields, helping to extract feature motifs. It convolves images through element-wise operations between the image and corresponding elements of the receptive field [22]. In our implementation, we used a two-dimensional convolutional kernel of size 3x3 to filter the input image. The first convolutional layer in our network learns 64 filters and the second one learns 128 filters.

The activation function for a convolved feature map is defined as $T_l^k = g_a(F_l^k)$ where F_l^k is the convolution output of the lth layer for the kth input feature map, g_a is the activation function and T_l^k is the transformed output. We use a ReLU activation function which removes negative values from our activation map, and helps train faster without any major penalty to generalization accuracy [22].

The max pooling layer is used to down sample the spatial dimensions of the convolved output by combining similar information in the neighborhood of the receptive field, and outputting the majority response using $Z_l^k = g_p(F_l^k)$, where F_l^k is the input feature map, g_a is the pooling function and Z_l^k is the pooled feature map. We operate the pooling function over a 2x2 square dimension, from which it returns the maximum value among the 4 values in the square matrix [22].

Dropout improves generalization by randomly setting input units or connections to 0 at a predefined frequency at each step during training time. We used it with a frequency of 0.1 after the first fully connected layer.

We added a layer that flattens the input structure to create a single long feature vector to be used by the dense layer for the final classification.

Dense layers are fully connected layers in which every neuron is connected to all neurons in the preceding and following layer. They globally analyze the output of all preceding layers and make a non-linear combination of selected features. We employ them at the output of the neural network with 3 neurons as per our classification task, with softmax as the activation function to get the multi-class probabilities and cross entropy for computing loss.

D. Hybrid CNN-SVM and CNN-XGBoost

By itself, the shallow architecture of classifiers like SVM and XGBoost can present difficulties in learning deep features. However, they display good generalization ability and can be useful when working with small datasets. The CNN model can extract features at multiple scales from an input image by way of convolution, pooling, etc. and can concatenate the outputs into a single feature vector, which can then be used as an input for a dense layer. However, they require large amounts of data and are prone to overfitting.

As a potential solution, we propose the use of the base CNN from Section III.C to extract the most discriminating features from the input images, and replace the last fully-connected softmax layer with an SVM or XGBoost classifier, with configurations similar to those defined in Section III.B. The classifier will then be trained on the automatically extracted feature vectors, which are generally more efficient to work with, in contrast to hand-crafted features.

E. Models with and Without PCA

In Section III.A, we empirically determined an optimal number of PCs and converted the original pool of 16,384 features to 1,024 synthetic features. We hypothesize that using PCA-transformed features as inputs will decrease computation time, and either increase or not have a significant impact on training performance. We trained all baseline models, the base CNN architecture and the hybrid CNN architectures on these PCA-transformed inputs.

IV. EXPERIMENTAL SETUP AND RESULTS

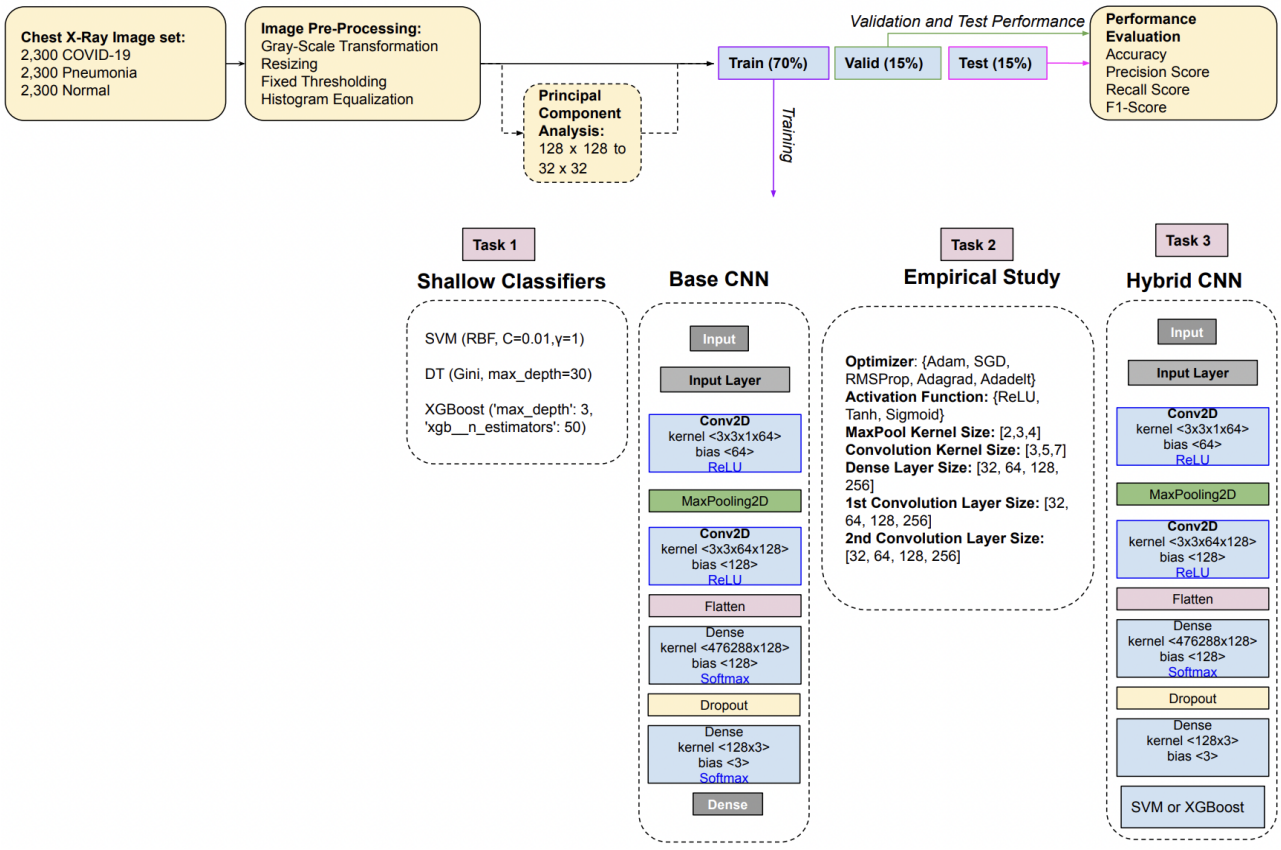

FIGURE 4: Experimental Setup. Task 1: Comparison of baseline models, with and without PCA, Task 2: Comparison of various CNN configurations, Task 3: Performance evaluation of the base CNN model and hybrid CNN approaches.

We split the original set of 6,939 images into 70% for training, 15% for validation and 15% for testing. All images were passed through the standard image pre-processing pipeline described in Section III.A. For various experiments, PCA was used to transform original images of size 128x128 into 32x32. To quantify classification performances, we used accuracy, precision, recall, F1 score as performance indices. Each model was trained 20 different times, and its mean test set performance is reported along with the 95% confidence interval.

Previously described models formed the basis of our experimental setup, and we performed three classification tasks as pictured in Figure 4 and described as follows:

1. ***Task 1: Comparison of baseline models, with and without PCA***

We compared the performance of 3 shallow classifiers: SVM (Kernel=RBF, C=0.01, Gamma=1), DT (Criterion='gini', Max. Depth=30), XGBoost (Max. Depth=3, Number of Estimators=50). We used cross validation grid search to train each model twice, once on the original feature set, and another time on the PCA transformed feature set. The results for this task are presented in Table 1.

TABLE 1: Performance evaluation of three shallow classifiers for multi-class classification.

	Accuracy		**Precision**		**Recall**		**F1**	
	No PCA	*PCA*	*No PCA*	*PCA*	*No PCA*	*PCA*	*No PCA*	*PCA*
SVM	0.982 ± 1.54e-16	0.977 ± 0.0	0.972 ± 1.541e-16	0.965 ± 1.541e-16	0.967 ± 0.0	0.966 ± 1.541e-16	0.970 ± 1.541e-16	0.965 ± 1.541e-16
XGBoost	0.940 ± 1.541e-16	0.950 ± 0.0	0.976 ± 1.541e-16	0.970 ± 0.0	0.841 ± 1.541e-16	0.880 ± 1.541e-16	0.893 ± 0.0	0.914 ± 0.0
DT	0.859 ± 0.00811	0.886 ± 0.004	0.753 ± 0.0181	0.816 ± 0.008	0.747 ± 0.0236	0.803 ± 0.006	0.742 ± 0.0132	0.803 ± 0.009

2. ***Task 2: Comparison of various CNN configurations***

An optimal CNN architecture can be built by iteratively tuning hyperparameters of the network and observing its performance on the dataset. In this task, each hyperparameter was selected from a distinct discrete or continuous domain and the performance of the resulting architecture was evaluated. We studied the effect of changing the hyperparameters, one at a time, to determine the sensitivity of the model to each hyperparameter.

The hyperparameter search space we experimented with included the optimizer, activation function, max. pool kernel size, convolution kernel size, the number of neurons in the dense layer, and the number of filters in the first and second convolution layers. All networks in this task used a final softmax layer with 3 neurons to predict the 3 classes. Optimal hyperparameter value in each iteration was determined based on model performance metrics. The hyperparameter space to be searched had 5 (optimizers) + 5 (dropout rates) + 3 (activation functions) + 3 (pool kernel sizes) + 3 (convolution kernel sizes) + 4 (dense layer sizes) + 3 (convolution layer 1 sizes) + 3 (convolution layer 2 sizes) = 30 hyperparameters. Consequently, 30 CNNs of varying configurations were trained in this task, the results for which are summarized in Tables A1-8 (Section Appendix A). Figure 5 and Table 2 detail the search space for each hyperparameter, default values used to start the search and the optimal parameter values, as determined by model performance.

TABLE 2: Hyperparameter search space and optimal values as determined by model performance.

Parameter	**Search Space**	**Default Value**	**Optimal Value**
Optimizer	['Adam', 'SGD', 'RMSProp', 'AdaGrad', 'AdaDelta']	Adam	Adam
Dropout Rates	[0.1, 0.2, 0.3, 0.4, 0.5]	0.1	0.3
Activation Function	['ReLU', 'Tanh', 'Sigmoid']	ReLU	ReLU
Pool Kernel Size	[2,3,4]	2	4
Convolution Kernel Size	[3, 5, 7]	3	5
Dense Layer Size	[32, 64, 128, 256]	128	64
Convolution Layer 1 Size	[32, 64, 128]	64	128
Convolution Layer 2 Size	[32, 64, 128, 256]	128	64

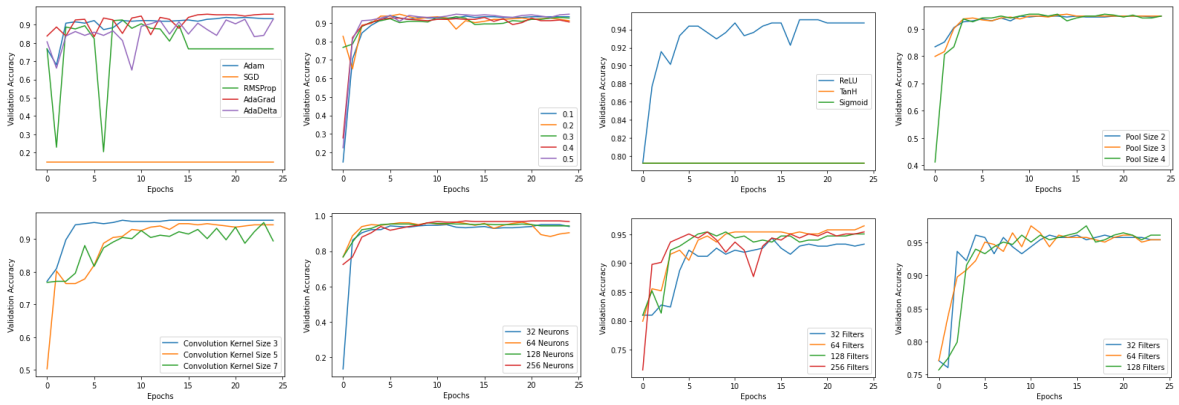

FIGURE 5: Results for comparison of various CNN configurations. A. Optimizer. B. Dropout rate. C. Activation function. D. Kernel pool size. E. Convolution kernel size. F. No. of neurons in the Dense layer. G. No. of Filters in Convolution layer 2. H. No. of Filters Convolution layer 1.

3. *Task 3: Classification of CNN features using shallow classifiers, with and without PCA*

From Figure 5, we noticed that in most cases, the training accuracy saturates at 10 epochs. Hence, we chose 10 as an optimal number of epochs to avoid overfitting, while still allowing the training process to converge. For all CNN models in this task, mini-batch training was used, with 256 input samples per batch. The train validation split was taken to be a conventional 80-20 for the training with 10 epochs. These mini-batches were selected at random and with replacement from the sample set. We used the built-in categorical cross-entropy as the loss function to be minimized and Adam as the optimizer.

We trained each model twice, once on the original feature set, and another time on the PCA-transformed feature set. For hybrid CNN models, the penultimate layer of a trained CNN was used as a feature input vector to the SVM or XGBoost classifiers. The results for this task are presented in Table 3.

TABLE 3: Performance evaluation of the base CNN model and hybrid CNN approaches.

	Accuracy		**Precision**		**Recall**		**F1**	
	No PCA	*PCA*	*No PCA*	*PCA*	*No PCA*	*PCA*	*No PCA*	*PCA*
Base CNN	0.928 ± 0.002	0.917 ± 0.002	0.928 ± 0.001	0.887 ± 0.006	0.815 ± 0.005	0.843 ± 0.004	0.861 ± 0.004	0.861 ± 0.003
Base CNN + SVM	0.930 ± 5.331e-17	0.899 ± 1.541e-16	0.928 ± 1.066e-16	0.872 ± 1.541e-16	0.814 ± 5.331e-17	0.785 ± 0.0	0.861 ± 5.331e-17	0.811 ± 0.0
Base CNN + XGBoost	0.929 ± 5.331e-17	0.897 ± 0.0	0.901 ± 0.0	0.845 ± 0.0	0.839 ± 0.0	0.804 ± 0.0	0.867 ± 0.0	0.812 ± 1.541e-16

V. DISCUSSION

Among the three shallow classifiers, our initial expectations were that XGBoost would produce better results by boosting the model during training by sequentially converting the weak classifier into a stronger one, but this was not the case. With the exception of precision, where XGBoost scored higher, SVM had the best performance across all metrics in both PCA and non-PCA subspaces. As expected, DT had the lowest performance, with a difference of 13% and 9% in non-PCA and PCA subspaces in comparison to SVM. Interestingly, SVM appears to be the most sensitive to PCA compression, as the accuracy, precision, recall and F1 scores all decreased in this subspace, suggesting dimension reduction was not effective for this model. In contrast, the metric scores of both XGBoost and DT increased when trained with the PCA-transformed data. However, despite the improvement in performance, XGBoost and DT still performed worse than SVM overall.

After testing the effect of different optimizers on the base CNN performance, we found that the Adam optimizer performed the best across all metrics. This was expected due to Adam's ability to adapt the learning rate scale for different layers and converge faster to find a critical point. Previous studies have also reported Adam's superior performance in comparison to other optimizers, further supporting our results.

We evaluated the performance of various dropout values ranging from 10% to 50% and found that a dropout of 30% yielded the best results. By dropping an increasingly larger number of units from the neural network, the model was better able to prevent overfitting, resulting in improved metrics; however, after the 30%

mark, the metrics began decreasing, indicating that too many units were being removed from the network. Therefore, we report 30% as the highest-performing dropout value for our 2D-CNN model.

From our experiments on activation functions, ReLU performed the best with all of its metrics beating both tanh and sigmoid by a significant margin. Owing to its non-linearity property, we expected similar results, keeping in mind the various studies that have been done comparing the different activations on similar image classification tasks.

Max pooling kernel size of 4 was the largest that we experimented with, and interestingly, we found that it also outperformed the other two lower sizes. Since it essentially partitions the input image into a set of non-overlapping regions, we believe that since the X-Ray images consisted of only 3 classes and were more alike in composition, a larger intentional down sampling on the input actually helped in outputting accurate values for the next convolution layers from larger grid sizes.

We found that 5x5 kernel sizes worked best, which are in between the highest and the lowest sizes we experimented with. It seems that the information available in the low level features present in local pixels are more diligently captured from a medium sized kernel than a low 3x3. At the same time, it also makes sense that a larger 7x7 kernel ends up losing valuable information by overlooking local pixels and skipping essential information captured successfully in the lower sized kernels, hence performing the worst among the three.

Amongst dense layer sizes ranging from 32-256, we found that a size of 64 had the highest performance in terms of accuracy, recall and F1 scores. After this point, the performance decreased with higher sizes across all metrics, except precision, which interestingly increased by 0.005. However, due to the overall improvement in performance at 64, we report that this dense layer size yielded the best results. Since dense layers are used to change dimensions of the output vector at each layer, this can lead to increases in performance, which we observed in our experiment. More filters in the input convolution layer gave better results which is similar to what we expected, as shown from our reported metrics. Convolution layer 1 with 128 filters was the highest we experimented with, and it ended up performing better than the other two lesser ones.

The highest performing conv layer 2 size was 64 across all metrics except precision, where the 256 size performed slightly better. At a feature layer size of 128, all metric scores diminished in comparison to the 64 size, but interestingly, the performance did not continue to decrease; instead, it improved at the 256 feature layer size. However, in terms of overall performance, the 64 feature layer size had the best results. We expected that the increase in number of filters would yield better results, which was observed in our experiment.

In terms of the base CNN and hybrid CNN models, we found that the base CNN+SVM model with non-PCA-transformed data performed the best. In the non-PCA subspaces, the base CNN+SVM model had the highest accuracy at 93%, but the base CNN had an accuracy of 92.8%, and the two models had very comparable results across all the metrics, with only differences ranging from 0-0.002%. There was a roughly 2% difference in precision and recall scores for CNN+XGBoost in comparison to the other two models, but otherwise a similar accuracy at 92.9%. This suggests replacing the last fully-connected softmax layer in the base CNN model with the shallow SVM and XGBoost classifiers had minimal impact on performance. In the PCA subspaces, the base CNN model had the highest scores across all metrics. However, between the PCA and non-PCA results, accuracy and precision for the base CNN model decreased, while recall increased by 3% and F1 scores stayed the same. For both CNN+SVM and CNN+XGBoost, all metric scores decreased in the PCA subspaces in comparison to non-PCA, illustrating that dimensionality reduction did not improve performance.

Based on the overall performances of the shallow and deep classifiers, we report that the shallow, non-PCA-transformed SVM classifier had the highest performance, with an accuracy of 98.2%. This was surprising, as we expected one of the hybrid CNN models to perform the best. However, we suspect that the SVM classifier may have overfit the data, which would explain its high metric scores; if we had used more than 10 epochs during training, we believe that the model would have achieved higher accuracy, which reinforces our notion of overfitting.

VI. CONCLUSION

In this study, we performed experiments on a Kaggle dataset composed of chest X-Rays from nine different data sources. We used a larger and more balanced COVID-19 image set in comparison to what has been reported in most previous research studies. We demonstrated the impact of image processing and dimensionality reduction techniques like PCA on subsequent model performance. We performed a comparative study involving three shallow classifiers (SVM, DT, and XGBoost), various 2D-CNNs configurations and hybrid CNN methods like CNN-XGBoost and CNN-SVM. Through proposed frameworks, we demonstrated that by applying shallow and deep learning techniques on chest X-Ray images, we can obtain reliable results in predicting COVID-19.

VII. FUTURE DIRECTIONS

In the future, we intend to experiment with additional dimensionality reduction techniques like Kernel PCA, image preprocessing techniques like Canny edge detection or the Gaussian Gradient operator, as well as image augmentation techniques like horizontal/vertical shifts, random rotations, zooms and brightness adjustments, to increase the volume of the data. This would help reduce the generalization error of the network. In Task 3, we used the penultimate layer of the trained base CNN model as input feature vectors to binary classifiers like CNN and XGBoost. In future iterations, we can extend the proposed hybrid CNN architectures to extract features from the fully connected layers of off-the-shelf/pre-trained networks like VGG16, GoogleNet, InceptionV3 and ShuffleNet, as described in some related works. In Task 2 of this study, we conducted a thorough hyperparameter space search. We noticed that the evaluation of many hyperparameter configurations can be subjective and time-consuming. Hyperparameter search can get prohibitively expensive as the number of parameters increases. Additionally, since hyperparameter values depend on each other, independently tuning the values does not lead to an overall optimal set of hyperparameter values. In future iterations of this experiment, we intend to iteratively build upon the most optimal architectures and look into mechanisms of narrowing the scope of the search through methods like random search, where we can randomly sample values from a bounded hyperparameter domain and make the computation time more manageable.

VIII. CONTRIBUTIONS

A. *Kopal Garg*

For our proposal, I conducted an extensive literature review on existing X-Ray image classification studies (**Section II**). I experimented with several image pre-processing steps (detailed in **Image_Preprocessing.ipynb**), and wrote the final pipeline which included PCA, grayscale transformation, resizing, fixed-binary thresholding, and histogram equalization (**Figures 1,2 and 3 and Section III.A**). I worked on all 3 baseline models (**PCA_XRay_Classification.ipynb, Non_PCA_XRay_Classification.ipynb**), conducting 10-fold cross validation grid searches on different sets of hyperparameters. I explained the rationale behind choosing each baseline model (**SVM, DT and XGBoost**), and each hyperparameter set in **Section III. B**, and included results for the experiments I conducted in **Table 1**, where I summarized 20 runs of each experiment using the mean and the 95% confidence interval (standard error of the mean). I worked on the PCA-vs-Non-PCA analysis for **Tasks 1 and 2**, and again summarized 20 runs of each experiment in **Table 3**, and wrote the Task 1 and Task 2 sections in **Section IV**, describing my work. I also worked on both the code and the discussion around the base CNN architecture and the hybrid CNN architectures (**Sections II C, D, E**, and **PCA_XRay_Classification.ipynb, Non_PCA_XRay_Classification.ipynb**). I wrote the **Conclusion** section, as well as the entire **Future Directions** section. I also helped compile the **References**, and did the final report formatting, as per the IEEE guidelines. This was a great learning experience (in several regards).

B. *Rohan Deepak Ajwani*

As described in Section IV Task 2, I conducted an **empirical study** where I built and trained **30 CNN models** with various **hyperparameter configurations** as outlined in Table 2. The hyperparameters chosen for the study included the optimizer, activation function, number of nodes in the dense layer, number of filters in the two convolutional layers and the kernel size for the pooling and convolutional layers. This involved extensive tuning of the hyperparameters of the network and observing the change in its performance on the dataset. In order to adhere to good statistical standards, I repeated each experiment **20 times**, to get a good estimate of the **mean and the 95% confidence interval** of the metrics, and presented my results in Tables A 1- 8 in the Appendix. I also plotted the **graphs** to illustrate the change in validation accuracy with the epochs while training the different CNN models (Figure 5). My code for these experiments is presented in **Empirical_Study_with_CNN.ipynb** as well as **Empirical_Study_with_CNN_Graphs.ipynb**. Towards the report, I described the methodology used for the empirical study. I also tabulated the results and represented them graphically, and also helped with the discussion. I brainstormed and helped implement the Hybrid CNN models described in Section III D and Section IV Task 3. I also experimented with model-agnostic explainability methods like SHAP, but due to time constraints, this wasn't presented as a part of our report.

C. *Yash Prakash*

I read and researched several **Datasets** for our image classification tasks online, compiled a list of them to use for our purposes and then wrote about them in our proposal and this report. I also worked on writing about the **background** for our project as well as our **motivation** in choosing the topics.

Regarding the code, I started out first from our team with several implementations of **base CNN neural networks in Keras (as seen in the non-PCA notebook)**, upon which my teammates and I decided to pick one implementation and built further on it. Upon selecting our base CNN model, I then helped conduct some experiments with how to go about building shallow classifiers and use scikit-learn algorithms with the Keras CNN models. For our Hyperparameter search, I also helped on all baseline as well as shallow models to use scikit-learn's **GridSearch** with it, and upon some research and several experiments, I helped experiment and brainstorm to come up with the best parameter to use in the grid to experiment with in the shallow as well as hybrid classifiers.

For the report, I worked on the sections of **background** which contained our **abstract** and **motivation** for the project. I then helped to come up with the experiments to conduct for image preprocessing which was then later implemented by our other teammate. I wrote about the **baseline CNN** implementation, built the baseline CNN and **shallow portions** of the **figure 4** demonstrating both neural nets architectures in detail. I then helped out the **Hybrid CNN-SVM** section, while also writing about the **tables** A3, A4, A5, A6, A8 in the Discussion sections. I also discussed and brainstormed how to demonstrate model explainability for our shallow and hybrid models, but we settled on simply displaying the overall architecture of our classifiers as a diagram.

D. *Charita Koya*

At the outset of the project, I researched COVID-19- and pneumonia-related chest X-Ray datasets, as well as numerous articles relating to image classification. These were included in the **Literature Review** in the proposal and **Section II (Related Works)** in this report. Alongside my teammates, I brainstormed and discussed potential shallow and deep classifiers to be used in this project, as well as hyperparameters to be tested and the overall architectural approach. Regarding the code, I worked on an iteration of the hybrid **CNN-SVM** model, as well as reviewed the code and results of the models that my teammates had built. In the report, I worked on **Section III.B** to explain the functionality and theory behind the **XGBoost** section. I also worked on **Task 2 (Empirical Study)** in **Figure 4** to detail the hyperparameters we were testing in the baseline CNN model. I wrote about tables 1 and 3, as well as A1, A2, A6 and A8 in **Section V (Discussion)** and explained the rationale for the results of each hyperparameter, as well as detailing what we expected to occur. I also reviewed the discussion regarding the other tables, contributed to **Section VII (Future Directions)** and compiled the **References** section.

REFERENCES

[1] B. Sekeroglu and I. Ozsahin, "Detection of COVID-19 from Chest X-Ray Images Using Convolutional Neural Networks", *SLAS TECHNOLOGY: Translating Life Sciences Innovation*, vol. 25, no. 6, pp. 553-565, 2020. Available: 10.1177/2472630320958376.

[2] M. Satu et al., "Convolutional Neural Network Model to Detect COVID-19 Patients Utilizing Chest X-ray Images", *medRxiv*, 2020. Available: 10.1101/2020.06.07.20124594.

[3] G. Jia, H. Lam and Y. Xu, "Classification of COVID-19 chest X-Ray and CT images using a type of dynamic CNN modification method", *Computers in Biology and Medicine*, vol. 134, p. 104425, 2021. Available: 10.1016/j.compbiomed.2021.104425.

[4] S. Minaee, R. Kafieh, M. Sonka, S. Yazdani and G. Jamalipour Soufi, "Deep-COVID: Predicting COVID-19 from chest X-ray images using deep transfer learning", *Medical Image Analysis*, vol. 65, p. 101794, 2020. Available: 10.1016/j.media.2020.101794.

[5] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition", *arXiv*, 2015. Available: https://arxiv.org/abs/1512.03385.

[6] "RESNET", *Pytorch.org*. [Online]. Available: https://pytorch.org/hub/pytorch_vision_resnet/. [Accessed: 15-Dec-2021].

[7] F. Iandola, S. Han, M. Moskewicz, A. Khalid, W. Dally and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size", *arXiv*, 2016. Available: https://arxiv.org/abs/1602.07360. [Accessed 15-Dec-2021].

[8] G. Huang, Z. Liu, L. Van Der Maaten and K. Q. Weinberger, "Densely Connected Convolutional Networks," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 2261-2269, doi: 10.1109/CVPR.2017.243.

[9] P. Sethy, S. Behera, P. Ratha and P. Biswas, "Detection of coronavirus Disease (COVID-19) based on Deep Features and Support Vector Machine", *International Journal of Mathematical, Engineering and Management Sciences*, vol. 5, no. 4, pp. 643-651, 2020. Available: 10.33889/ijmems.2020.5.4.052.

[10] J. Stubblefield et al., "Transfer learning with chest X-rays for ER patient classification", *Scientific Reports*, vol. 10, no. 1, 2020. Available: 10.1038/s41598-020-78060-4.

[11] Rekha Rajagopal, "Comparative Analysis of COVID-19 X-ray Images Classification Using Convolutional Neural Network, Transfer Learning, and Machine Learning Classifiers Using Deep Features", *Pattern Recognition and Image Analysis*, vol. 31, no. 2, pp. 313-322, 2021. Available: 10.1134/s1054661821020140.

[12] A. Zargari Khuzani, M. Heidari and S. Shariati, "COVID-Classifier: an automated machine learning model to assist in the diagnosis of COVID-19 infection in chest X-ray images", *Scientific Reports*, vol. 11, no. 1, 2021. Available: 10.1038/s41598-021-88807-2.

[13] J. Cohen, P. Morrison and L. Dao, "COVID-19 image data collection", *GitHub*, 2020. [Online]. Available: https://github.com/ieee8023/covid-chestxray-dataset. [Accessed: 15-Dec-2021].

[14] L. Wang et al., "Figure 1 COVID-19 Chest X-ray Dataset Initiative", *GitHub*, 2020. [Online]. Available: https://github.com/agchung/Figure1-COVID-chestxray-dataset. [Accessed: 15-Dec-2021].

[15] A. Haghanifar, M. Molahasani Majdabadiand S. Ko, “COVID-19 Chest X-Ray Image Repository”. figshare, 2020, doi: 10.6084/m9.figshare.12580328.v2.

[16] H. B. Winther, H. Laser, S. Gerbel, S. Maschke, J. Hinrichs, J. Vogel-Claussen, F. Wacker, M. Höper, B. C. Meyer, “COVID-19 Image Repository”. figshare, 12-May-2020, doi: 10.6084/m9.figshare.12275009.v1.

[17] Alqudah, Ali Mohammad; Qazan, Shoroq (2020), “Augmented COVID-19 X-ray Images Dataset”, Mendeley Data, V4, doi: 10.17632/2fxz4px6d8.4

[18] P. Mooney, "Chest X-Ray Images (Pneumonia)", *Kaggle.com*, 2017. [Online]. Available: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia. [Accessed: 15-Dec-2021].

[19] "NIH Chest X-rays", *Kaggle.com*, 2021. [Online]. Available: https://www.kaggle.com/nih-chest-xrays/data. [Accessed: 15-Dec-2021].

[20] Chih-Wei Hsu and Chih-Jen Lin, "A comparison of methods for multiclass support vector machines," in *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415-425, March 2002, doi: 10.1109/72.991427.

[21] "1.12. Multiclass and multi output algorithms", *scikit-learn*, 2021. [Online]. Available: https://scikit-learn.org/stable/modules/multiclass.html. [Accessed: 15-Dec-2021].

[22] A. Khan, A. Sohail, U. Zahoora and A. Qureshi, "A survey of the recent architectures of deep convolutional neural networks", *Artificial Intelligence Review*, vol. 53, no. 8, pp. 5455-5516, 2020. Available: 10.1007/s10462-020-09825-6 [Accessed 15 December 2021].

APPENDIX

A. Task 3: Comparison of various CNN configurations

TABLE A1. Performance evaluation of different optimizers on the base 2D-CNN model.

Optimizer	Accuracy	Precision	Recall	F1
Adam	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
SGD	0.2050 ± 0.0931	0.0683 ± 0.0310	0.3333 ± 2e-17	0.1032 ± 0.0304
RMSProp	0.9341 ± 0.0099	0.9072 ± 0.0158	0.8738 ± 0.0205	0.8850 ± 0.0134
Adagrad	0.8879 ± 0.0200	0.8548 ± 0.0607	0.7726 ± 0.0581	0.7796 ± 0.0537
Adadelta	0.8761 ± 0.0188	0.8447 ± 0.0468	0.7353 ± 0.0678	0.7395 ± 0.0626

TABLE A2. Performance evaluation of different dropout rates on the base 2D-CNN model.

Dropout	Accuracy	Precision	Recall	F1
10%	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
20%	0.9540 ± 0.0034	0.9400 ± 0.0078	0.8937 ± 0.0070	0.9149 ± 0.0055
30%	0.9578 ± 0.0032	0.9401 ± 0.0078	0.9072 ± 0.0084	0.9225 ± 0.0061
40%	0.9525 ± 0.0026	0.9309 ± 0.0070	0.9029 ± 0.0103	0.9154 ± 0.0049
50%	0.9417 ± 0.0329	0.9276 ± 0.0336	0.9016 ± 0.0100	0.9072 ± 0.0320

TABLE A3. Performance evaluation of different activation functions on the base 2D-CNN model.

Activation Function	Accuracy	Precision	Recall	F1
ReLU	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
TanH	0.7869 ± 5e-17	0.2623 ± 0.0	0.3333 ± 2e-17	0.2935 ± 2e-17
Sigmoid	0.8783 ± 0.0277	0.7492 ± 0.1160	0.6685 ± 0.0929	0.6750 ± 0.1052

TABLE A4. Performance evaluation of different max pooling layer kernel sizes on the base 2D-CNN model.

Pool Kernel Size	Accuracy	Precision	Recall	F1
2	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
3	0.9566 ± 0.0039	0.9384 ± 0.0063	0.9010 ± 0.0125	0.9181 ± 0.0077
4	0.9651 ± 0.0044	0.9436 ± 0.0076	0.9346 ± 0.0117	0.9381 ± 0.0077

TABLE A5. Performance evaluation of different convolution kernel sizes on the base 2D-CNN model.

Conv. Kernel Size	Accuracy	Precision	Recall	F1
3	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
5	0.9496 ± 0.0085	0.9282 ± 0.0127	0.8909 ± 0.0227	0.9055 ± 0.0178
7	0.9399 ± 0.0163	0.8957 ± 0.0465	0.8819 ± 0.0338	0.8843 ± 0.0393

TABLE A6. Performance evaluation of different dense layer sizes on the base 2D-CNN model.

Dense Layer Size	Accuracy	Precision	Recall	F1
32	0.8783 ± 0.0277	0.7492 ± 0.1160	0.6685 ± 0.0929	0.6750 ± 0.1052
64	0.9503 ± 0.0033	0.9161 ± 0.0101	0.9067 ± 0.0073	0.9103 ± 0.0051
128	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
256	0.9463 ± 0.0178	0.9045 ± 0.0711	0.8739 ± 0.0600	0.8878 ± 0.06568

TABLE A7. Performance evaluation of different convolution layer 1 sizes on the base 2D-CNN model.

Conv. Layer 1 Size	Accuracy	Precision	Recall	F1
32	0.9459 ± 0.0037	0.9292 ± 0.0077	0.8755 ± 0.0097	0.8996 ± 0.0077
64	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
128	0.9506 ± 0.0035	0.9290 ± 0.0058	0.8872 ± 0.0107	0.9062 ± 0.0064

TABLE A8. Performance evaluation of different convolution layer 2 sizes on the base 2D-CNN model.

Conv. Layer 2 Size	Accuracy	Precision	Recall	F1
32	0.9283 ± 0.0105	0.9006 ± 0.0161	0.8371 ± 0.0259	0.8649 ± 0.0223
64	0.9535 ± 0.0029	0.9323 ± 0.0085	0.9043 ± 0.0059	0.9172 ± 0.0042
128	0.9474 ± 0.0045	0.9211 ± 0.0071	0.8908 ± 0.0111	0.9044 ± 0.0079
256	0.9517 ± 0.0076	0.9343 ± 0.0066	0.8941 ± 0.0230	0.9104 ± 0.0166