Deep Learning- Final Course Project

Asaf Mesilaty

Submitted as final project report for the DL course, BIU, 2024

1 Introduction

This final project for the Deep Learning course explores the classification of chest X-ray images into categories of healthy, viral pneumonia, and bacterial pneumonia using deep learning techniques within the constraints of limited computational resources. Utilizing a dataset sourced from Kaggle [1], consisting of 5,863 chest X-ray images, the project incorporates binary and categorical classification models, anomaly detection with autoencoders, and model explainability through Heat Map by Occlusion and Grad-CAM, providing a comprehensive learning experience.

Figure 1: Sample X-ray images.

1.1 Related Works

The implementation of Grad-CAM presented here is inspired by an example provided in the Keras documentation [2], laying the foundation for understanding our models' decisions.

2 Solution

2.1 General Approach

This project explored building both binary and categorical models for image classification. To improve model interpretability, anomaly detection was incorporated using an autoencoder. To visualize the latent representations learned by these models, t-distributed Stochastic Neighbor Embedding (t-SNE) was employed to project them into a 2D space. Additionally, heatmaps generated using occlusion and Grad-CAM techniques were leveraged to further explain the models' decision-making processes.processes

2.2 Design

Models were developed using the Keras library, with visualization facilitated by matplotlib.pyplot. The ImageDataGenerator from Keras enabled training on randomized, rotated, flipped, and shifted images, enriching the models' learning process.

- Both Binary and Categorical models did not utilize validation techniques due to a handcrafted training process.
- Binary model: A combination of CNN and DNN layers with a sigmoid output optimized for distinguishing healthy from pneumonia cases.

```
binary_model = Sequential([
    Conv2D(16, (3, 3), activation='relu', input_shape=img_shape),
    Conv2D(16, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Conv2D(32, (3, 3), activation='relu', input_shape=img_shape),
    MaxPooling2D(2, 2),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Flatten(),

    Dense(64, activation='relu'),
    Dropout(0.5),

    Dense(1, activation='sigmoid')
])

binary_model.compile(optimizer=Adam(), loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 2: Binary model architecture.

- Categorical model: Adapted from the binary model through transfer learning, classifying images into three distinct classes.

```
base_model = Sequential(binary_model.layers[:-4])

categorical_model = Sequential([
    base_model,
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),

    Flatten(),

    Dense(64, activation='relu'),
    Dropout(0.5),

    Dense(3, activation='softmax')
])

categorical_model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 3: Categorical model architecture.

- Autoencoder: Utilized a small network architecture, focused on anomaly detection by learning normal image representations.

```
input_img = Input(shape=(img_height, img_width, 1))  # grayscale images

# Encoder
x = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(16, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
encoded = Dropout(0.2)(x)  # Adding dropout for regularization

# Decoder
x = Conv2D(16, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Dropout(0.2)(x)  # Adding dropout for regularization
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer=Adam(), loss='mean_squared_error')
```

Figure 4: Autoencoder model architecture.

You can see the MSE distribution and the 5% percentile threshold of the normal training set in figure 5.

- Adam optimizer was chosen for all models, outperforming SGD across various learning rates.

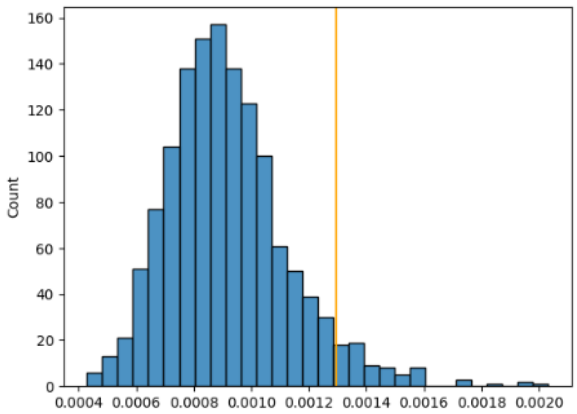

Figure 5: MSE distribution with threshold.

Theoretically, every hidden layer can represent an embedding layer. We can extract an output of any hidden layers and treat it as an embedding vector. Still, the point is not only to lower the input dimension but also to create a meaningful relationship between them.

The project investigated the use of K-Nearest Neighbors (KNN) on the embedding layer for further analysis. One example involved using the dense layer directly preceding the dropout layer in the binary model. The implemention was simple and efficient as seen in the figure:

```
[ ] class EmbeddingModel(Model):
        def predict_step(self, data):
            x, y = data
            return self(x, training=False), y
```

```
[ ] embedding_layer = binary_model.layers[-3]  # The Dense layer before Dropout
    embedding_model = EmbeddingModel(inputs=binary_model.input, outputs=embedding_layer.output)
```

```
[ ] train_embeddings, train_labels = embedding_model.predict(train_data)
    test_embeddings, test_labels = embedding_model.predict(test_data)
```

```
    325/325 [==============================] - 121s 370ms/step
    37/37 [==============================] - 12s 309ms/step
```

```
[ ] knn_classifier = KNeighborsClassifier(n_neighbors=5)
    knn_classifier.fit(train_embeddings, train_labels)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

Figure 6: KNN on the embedding layer of the binary model

Plotting the training and test data into 2 dimensional space using T-SNE gives us:

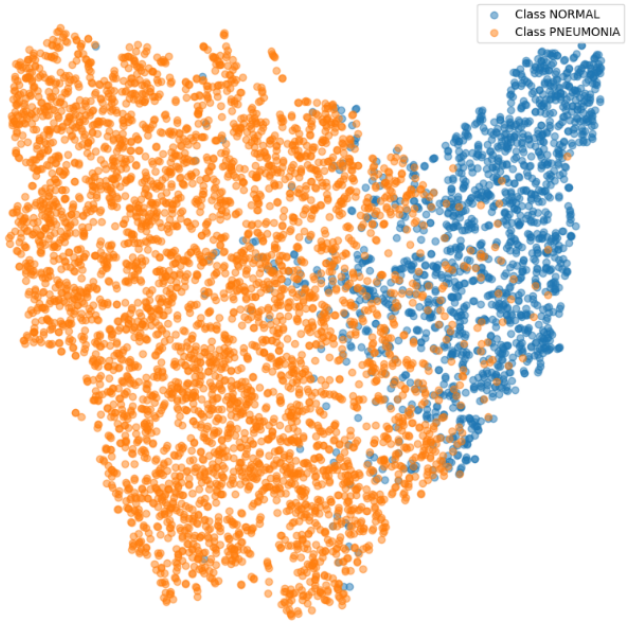

Figure 7: Binary model embedding T-SNE visualisation

3 Experimental Results

Experiments demonstrated the binary model's superior accuracy, while the categorical model faced challenges in distinguishing between viral and bacterial pneumonia, attributed to embedding similarities. Anomaly detection achieved modest success, indicating the potential for improvements with more complex models.

Model	Accuracy
Binary	0.85
Categorical	0.76
KNN on Binary's Embedding	0.78
KNN on Categorical's Embedding	0.7
Autoencoder (Threshold)	0.33
Autoencoder (KNN on Encoded)	0.66

Table 1: Model performance comparison.

Heat Map by Occlusion figures 8 and 9 show the lower middle part of the chest as the most sensitive area.

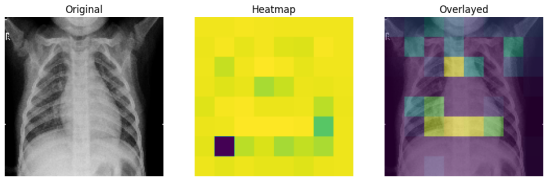

Figure 8: Normal Heat map by occlusion

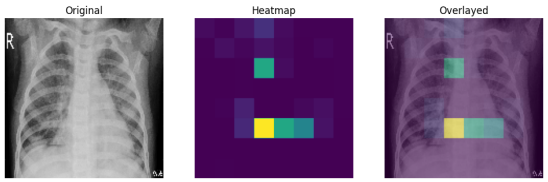

Figure 9: Pneumonia Heat map by occlusion

Grad-CAM visualizations in figure 10 highlight bone structures and, in sick images, the presence of the letter 'R'.

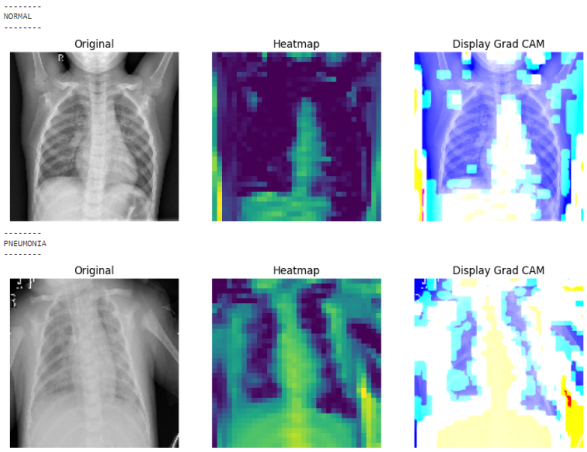

Figure 10: GRAD-CAM Heat map

4 Discussion

The binary model's performance, difficulties with the categorical model, and varying success in anomaly detection underline the challenges of medical image analysis with limited resources. The project's value lies in demonstrating deep learning techniques and model explainability, focusing on education over achieving state-of-the-art performance.
I strongly suggest reviewing the train.ipynb Colab Notebook For a more in-depth understanding of this project.

5 Code

The project is fully documented in Google Colab notebooks, accessible below:
Training: (Includes the entire project)train.ipynb Colab Notebook
Testing: test.ipynb Colab Notebook

References

[1] Kaggle Chest X-ray Pneumonia Dataset
[2]Grad-CAM class activation visualization