FUNDAMENTALS OF DEEP LEARNING

Final Project

Arunkkumar Karthikeyan

Priya Yadav

Maria Karakoulian

24 March, 2023

Exercise 1: Image Classification

PROJECT BACKGROUND

The objective of this exercise is to analyze the “Vehicles in Accidents” dataset using different Neural Network models, including Multi-Layer Perceptron Networks (MLP), Convolutional Neural Networks (CNN), and Transfer Learning. It aims to compare the performance of these models, identify the best architecture and parameters, and recommend the most suitable approach for detecting accidents in real-world scenarios. The goal is to detect the object and identify the vehicles belonging to one of the following categories: **“light vehicle”, “heavy vehicle”** or **“motorcycle”.**

The " Vehicles in Accidents" is a subset of the “Accident Images Analysis” dataset, a publicly available dataset that contains over 2,636 images of damaged vehicles, along with metadata. The images are labeled with information on the type of the vehicles. They have dimensions of 224x224 pixels and are in RGB color format, with three color channels.

DATA PREPARATION

In order to prepare our dataset of images for analysis, we used a Python function to load the images from our basepath and convert them into arrays. We specified the target size of the images as 224x224 pixels, which is a common size for image recognition models.

We then split the images into a training set and a test set, with 70% of the data going to the training set and 30% to the test set. This allows us to train our models on a portion of the data and test their accuracy on a separate portion.

To prepare our data for classification, we used the ***to_categorical*** function to convert our integer labels into one-hot encoded vectors. This means that each label is represented as a vector of length equal to the number of classes, with a value of 1 in the position corresponding to the correct class and 0's in all other positions.

This approach allows our classification models to predict the probability of each class and choose the class with the highest probability as the predicted label. By using one-hot encoded vectors, we can ensure that our model correctly assigns a probability to each class and does not favor any one class over another.

MULTI-LAYER PERCEPTRON NETWORKS (MLP)

A. Model Architecture

We implemented Multi-Layer Perceptron (MLP) networks for our accident detection dataset. We converted the images to grayscale and then scaled them with values between 0 and 1. We defined three different MLP models with varying architectures:

- MLP1 with 2 hidden layers of 64 neurons each
- MLP2 with 4 hidden layers of 128 neurons each
- MLP3 with 2 hidden layers of 256 neurons each

We compiled the MLP models using the ***categorical cross-entropy*** loss function and the ***Adam*** optimizer, which is an algorithm used for stochastic gradient descent.

All MLPs had a ***softmax*** activation function in the output layer, which helped us to obtain probability distributions over the three classes.

B. Training and Validation

We trained the models on the training dataset for 10 epochs with a batch size of 32 and obtained validation accuracy scores for each model. We used these scores to evaluate the performance of each model and choose the best one for our task.

C. Performance Evaluation and Comparison

After training, we evaluated the MLP models on a test set and observed the following results:

	Model	Train Acc.	Test Acc.	Train Loss	Test Loss
0	MLP1	0.615718	0.548673	0.853076	1.012708
1	MLP2	0.595122	0.527181	0.862685	0.958125
2	MLP3	0.637398	0.554994	0.824424	0.986209

MLP3 model, with two hidden layers of 256 neurons each, achieved the highest validation accuracy score of 0.55. This means that the MLP model can correctly classify images of vehicles involved in accidents in 5 out of 10 cases.

However, we should note that we might need to improve the model further, as the validation accuracy is still relatively low, indicating that the model might be overfitting the training data.
We can also see that as the model complexity increases (i.e., from MLP1 to MLP3), the training loss decreases, indicating that the model is learning better representations of the data. Yet, this

is not reflected in the validation loss, which remains relatively high. This indicates that the models might be overfitting the training data and not generalizing well to unseen data.

Finally, we can see that the test accuracy is lower than the training and validation accuracy for all the models. This might be expected since the models have not seen the test data during training and might not generalize well to it.

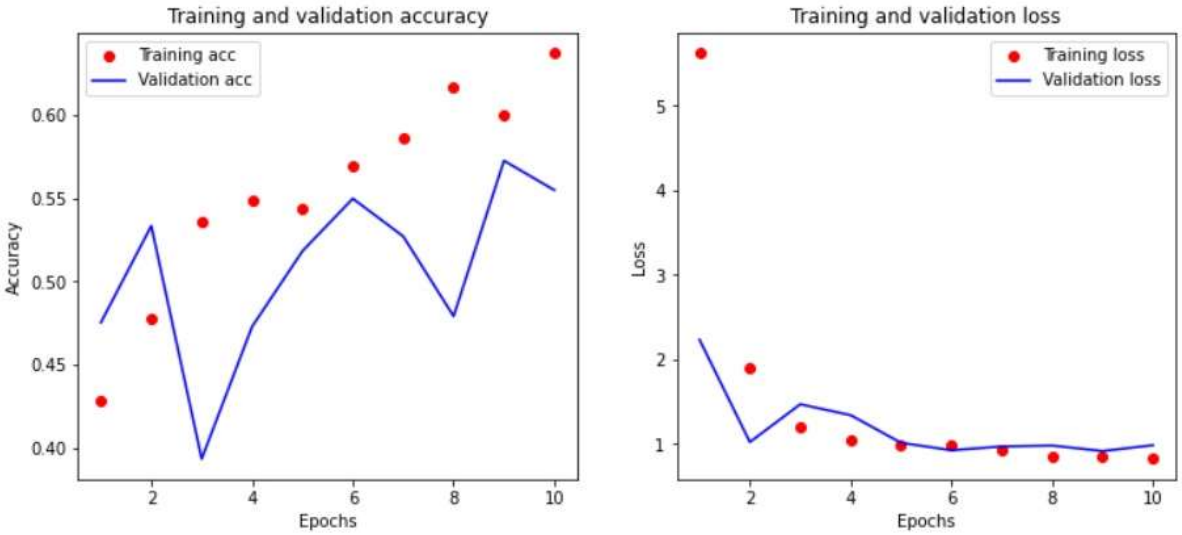


Fig 1. MLP3 Training and Test Loss and Accuracy Results Comparison

D. Discussion on the Choice of Architecture and Parameters

We experimented with three different architectures, with varying numbers of hidden layers and neurons per layer. We chose the number of neurons based on a trade-off between model complexity and accuracy. We wanted to find a balance between having enough neurons to capture complex relationships in the data, but not so many that the model becomes overfit to the training data and performs poorly on new data.

We also experimented with different batch sizes and numbers of epochs to find the optimal hyperparameters for each model.

We chose to use the ***ReLU*** activation function for the dense layers in all three networks because it is a commonly used activation function in neural networks that has been shown to work well for image classification tasks. We also used the ***softmax*** activation function for the output layer to ensure that the output of the network represented a probability distribution over the three classes.

We trained the networks using the ***categorical cross-entropy*** loss function and the ***Adam*** optimizer, which is an efficient and widely used algorithm for stochastic gradient descent.

CONVOLUTIONAL NEURAL NETWORKS (CNN)

A. Model Architecture

We implemented Convolutional Neural Networks (CNN) networks for our accident detection dataset. CNNs are a type of neural network particularly useful in image analysis because of their ability to capture spatial features.

The different layers used in the architecture:

- **Batch Normalization Layer**: This layer is used on the output of the previous convolutional layer to improve the training speed and stability of the model
- **Max Pooling Layer:** This layer is applied on the output of the previous batch normalization layer. This helps to reduce the spatial dimensions of the output and prevent overfitting
- **Global Average Pooling Layer:** This technique is a pooling operation which involves taking the average of all feature maps in the convolutional layer and returning a single value for each feature map. This is different from traditional pooling methods, which typically use max pooling or average pooling over small regions of the input
- **Dropout Layer:** This layer randomly sets a fraction of the input units to zero during training to prevent overfitting

The different parameters used in the architecture:

- **Padding:** This parameter is used to keep the size of the output feature map the same as the input feature map by adding extra pixels around the edges of the input feature map. When set to ***same,*** the input feature map is padded with zeros so that the output feature map has the same spatial dimensions as the input feature map.
- **Regularization (L2):** This method adds a penalty term to the loss function that is proportional to the square of the magnitude of the weight values. The purpose of this regularization is to encourage the model to learn smaller weights and reduce the effect of outlier

We defined six different CNN models with varying architectures:

- CNN1 with 2 convolutional layers of 32 and 64 neurons respectively, with padding set to same, 2 max pooling layers, and 1 dense layer of 64 neurons
- CNN2 with 2 convolutional layers of 32 and 64 neurons respectively without padding, 2 max pooling layers, and 1 dense layer of 64 neurons
- CNN3 with 3 convolutional layers of 32, 64 and 128 neurons respectively without padding, 3 max pooling layers, and 1 dense layer of 128 neurons

- CNN4 with 3 convolutional layers of 32, 64 and 128 neurons respectively without padding, 2 max pooling layers, global average pooling for the last convolutional layer
- CNN5 with 1 convolutional layer of 32 neurons, 1 max pooling layer, batch normalization, and 1 dense layer of 64 neurons
- CNN6 with 4 convolutional layers of 32, 64, 128, 256 neurons respectively, 4 max pooling layers, 5 dropout layers, and 2 dense layers

The models are compiled with ***categorical cross-entropy*** loss and ***Adam*** optimizer.

B. Training and Validation

We trained the models on the training dataset for 10 epochs with a batch size of 32 and obtained validation accuracy scores for each model. We used these scores to evaluate the performance of each model and choose the best one for our task.

C. Performance Evaluation and Comparison

After training, we evaluated the CNN models on a test set and observed the following results:

Model	Train Acc.	Test Acc.	Train Loss	Test Loss
CNN1	0.962060	0.634640	0.105097	2.927246
CNN2	0.997832	0.623262	0.015883	2.442960
CNN3	0.996748	0.625790	0.011998	2.766216
CNN4	0.557182	0.517067	0.928500	0.970866
CNN5	0.996748	0.711757	0.020613	1.014446
CNN6	0.340921	0.332491	1.098885	1.098848

From the above results of all CNN models, CNN5 has the highest test accuracy of 0.716, which is significantly higher than the other models. CNN5 includes a Batch Normalization layer after the first convolutional layer, which helps to normalize the input values and might have helped speed up the learning process. Additionally, it has only one convolutional layer, which makes it computationally efficient, while still performing well.

However, the training accuracy for CNN5 is very high (0.996), which could indicate overfitting. To reduce overfitting, we will be going to apply regularization techniques such as Dropout or L2 regularization.

D. Additional Techniques

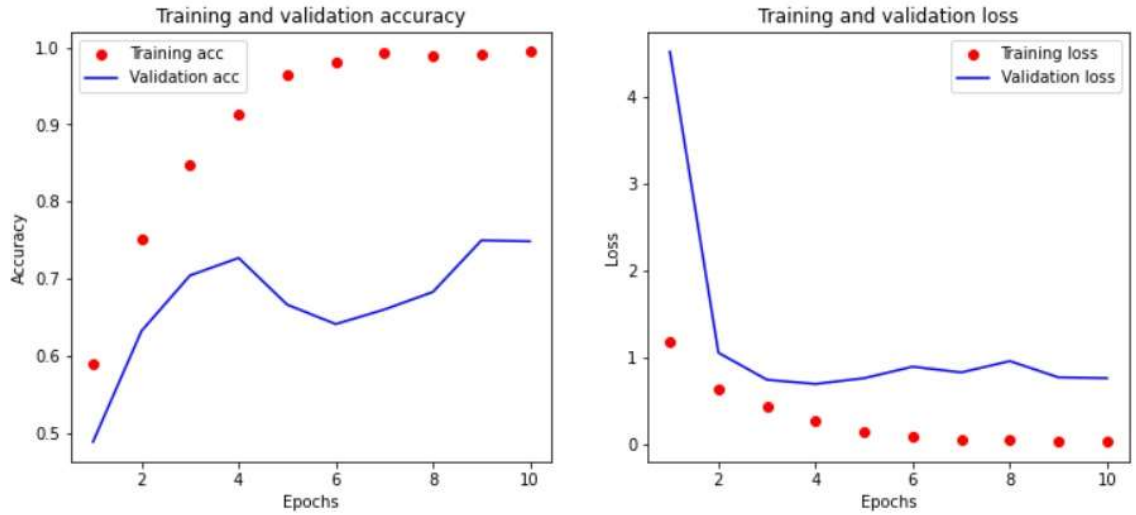


Fig 2. CNN5 Training and Test Loss and Accuracy Results Comparison

We implemented six additional models using batch normalization, padding and regularizations.

- CNN7 with 1 convolutional layer of 32 neurons, 2 batch normalization layers, 1 max pooling layer, a dense layer of 64 neuron, 1 dropout layer, and 1 output layer
- CNN8 with 8 convolutional layers of 2x 32, 2x 64, 2x 128, 2x 256 neurons respectively, 8 batch normalization layers, and 4 max pooling layers, 5 dropout layers and 1 dense layer of 512 neurons. The model also employs padding set to same and L2 regularization with a coefficient of 0.01 on each convolutional layer
- CNN9, has only 2 convolutional layers of 32 and 64 neurons, 3 batch normalization layers, 2 max pooling layers and 1 dense layer of 128 neurons. The model also employs L2 regularization with a coefficient of 0.5 on each convolutional layer
- CNN10 with 8 convolutional layers of 2x 64, 2x 128, 2x 256, 2 x 512 neurons respectively, 8 batch normalization layers, 4 max pooling layers, 5 dropout layers and 1 dense layer of 1024 neurons. The model also includes padding set to same and L2 regularization with a coefficient of 0.01 on each convolutional layer
- CNN11 with 8 convolutional layers of 2x 32, 2x 64, 2x 128, 2x 256 neurons respectively, 8 batch normalization layers, 4 max pooling layers, 5 dropout layers, and a dense layer of 512 neurons. The model also employs L2 regularization with a coefficient of 0.001 on each convolutional layer

The models are compiled with ***categorical cross-entropy*** loss, ***Adam*** optimizer, and ***Relu*** activation

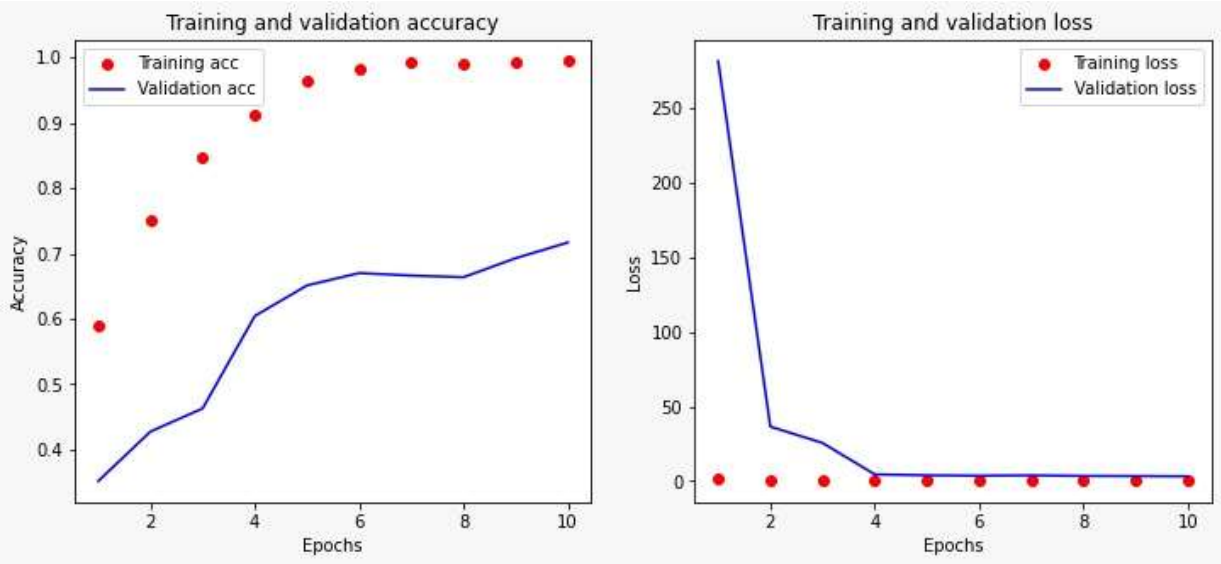


Fig 3. CNN11 Training and Test Loss and Accuracy Results Comparison

E. Discussion on the Choice of Architecture

From the results shown, it appears that CNN11 has achieved a considerably higher test accuracy of 0.716 compared to the other models. Although CNN11 has a more complex architecture, it has achieved the same accuracy as CNN5 without overfitting. However, it is worth noting that CNN11 may not be the most computationally efficient option.

Given the results, it may be more practical to choose CNN5 as our model and implement techniques such as data augmentation or transfer learning to improve its performance. These techniques can potentially enhance the ability of the model to generalize to new data while reducing the risk of overfitting. By doing so, we can achieve better accuracy while maintaining computational efficiency.

F. Data Augmentation

Data augmentation involves generating additional training data by applying various random transformations to the original images, such as rotation, scaling, and shearing. This helps to increase the size and diversity of the training set, which can lead to better generalization and improved accuracy.

Early stopping is a technique that involves monitoring the performance of the model during training and stopping the training process when the performance on a validation set stops improving. This helps to prevent overfitting, where the model becomes too specialized to the training data and performs poorly on new, unseen data.

To further improve the accuracy of our model, we used data augmentation and early stopping during training on our best performing model (CNN5). We applied 3 different optimizer techniques: ADAM, RMSPROP, and SGD, with 32 batch size and 10 epochs.

The results were as follows:

	Train acc.	Train Loss	Test Acc.	Test Loss
ADAM	0.7259	0.6419	0.7295	0.6505
RMSPROP	0.7683	0.5741	0.7358	0.7166
SGD	0.6646	0.7729	0.9093	0.6195

From the above we found out that the data augmentation with ***adam*** optimizer was giving the best result (test accuracy of 0.729). This is a slight improvement over our previous model, which had an accuracy of 0.71.

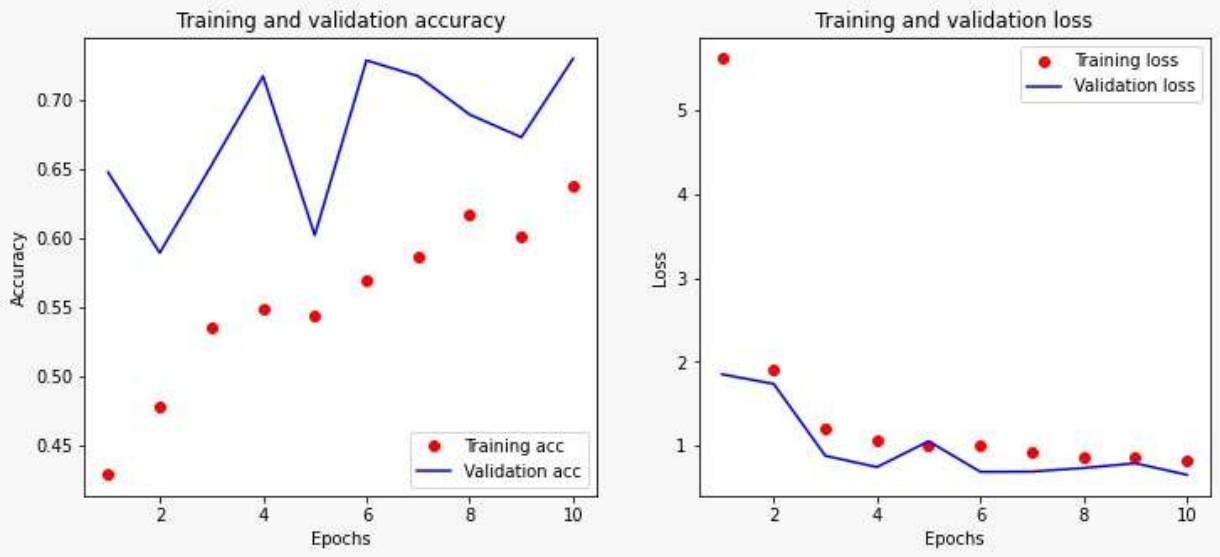


Fig 4. CNN5 with Data Augmentation Training and Test Loss and Accuracy Results Comparison

G. Interpretability Techniques

Interpretability techniques in deep learning involve methods to understand and explain how a neural network model is making its predictions. These techniques are important because deep learning models are often considered "black boxes," meaning that their internal workings are not easily understood by humans.

First we sampled our training set with 800 images and the selected 3 random images to explain. Then we used 2 interpretability techniques to understand which regions of our input images were most important for the CNN5 model's predictions:

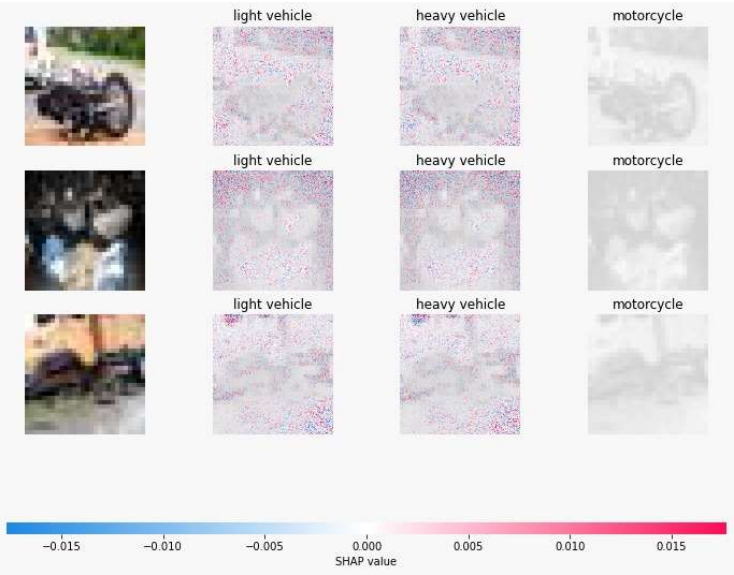
- **SHAP (SHapley Additive exPlanations)**: This technique is based on game theory and provides an explanation of the model's prediction for a particular instance by computing the contribution of each feature to the prediction. It helps to identify the most important features for the prediction and how each feature contributes to the final outcome.

We used SHAP to generate visual explanations of the top 3 predicted classes for a given image, and to show how much each pixel of the image contributes to the prediction of each class.

The first function ***map2layer*** is used to map the input data to the specified layer in the model. This is necessary because SHAP operates on intermediate layers of the model rather than the input or output layers.

Next, we randomly select 3 images from the training data and create a ***GradientExplainer*** object. This object takes the model layer (cnn5.layers[3].input) as input and the model layer (cnn5.layers[-1].output) as output. We also provided a mapping of the input data to the layer using the map2layer function.

Finally, we used the ***shap_values*** and indexes generated by the ***GradientExplainer*** object to create a plot of the SHAP values for the selected images. The shap_values are the contribution of each pixel to the output, and the indexes are the predicted classes for the selected images



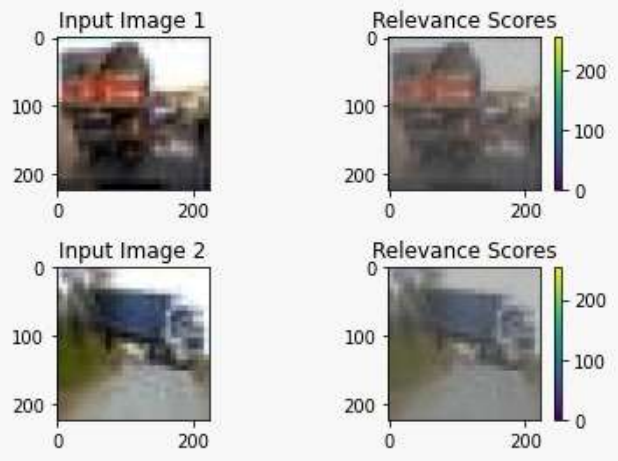
The ***shap.image_plot*** function then generates a grid of images as we can see on the right, where each row corresponds to a different input image and each column corresponds to a different predicted class. The images in the grid show the input image with an overlay of the SHAP values for each pixel. The intensity of the overlay color indicates the magnitude and direction of the feature's contribution to the prediction. Positive SHAP values are shown in red, while negative SHAP values are shown in blue. The intensity of the color indicates the magnitude of the SHAP value, with darker colors indicating larger magnitudes. By examining the SHAP plots, we can gain insight into which features of the input image are most important for the model's predictions. This can help us understand how the model is making its decisions and identify potential biases or issues with the model's training data.

- **LRP (Layer-wise Relevance Propagation):** This technique is used to interpret deep neural networks by assigning relevance scores to individual input features, such as pixels in an image. LRP works by propagating the output prediction back through the layers of the network to determine which features in the input data were most important in producing that prediction. There are several variations of LRP, each with different rules for propagating relevance scores through the layers of the network.

We used the ***alpha2beta1*** rule, which is a popular and effective method for analyzing convolutional neural networks. By applying LRP to a trained model and visualizing the resulting relevance scores, we can gain insights into how the model is making its predictions and identify any biases or errors that may be present. This can help improve the accuracy and fairness of the model, and can also aid in the development of more effective models.

This code demonstrates the use of Layer-wise Relevance Propagation (LRP) to interpret a pre-trained convolutional neural network model(our best model CN5). The code loads a pre-trained model and creates a new model that excludes the last ***Softmax*** layer. It then selects two random images from the training set and applies the LRP analyzer to generate relevance scores for each pixel. These scores are normalized between 0 and 1 and then plotted as an overlay on top of the original images.

By visualizing the relevance scores, we can gain insights into the most important features that the model is using to make its predictions. This can be helpful in identifying biases or errors in the model, as well as improving its accuracy and fairness.



The plot shows the original input image alongside the relevance scores, which provide an indication of how much each pixel in the input contributes to the output of the model. The darker the pixel in the relevance scores, the more important it is for the final prediction.

H. Transfer Learning

Transfer learning is a technique in machine learning where a pre-trained model is used as a starting point for a new task. In transfer learning, the pre-trained model's weights are used to initialize the new model, which is then trained on a new dataset for a different task. Transfer learning is useful when we have limited data for the new task, or when training a model from scratch would be computationally expensive.

Transfer learning has been successfully applied in many applications, including image classification, natural language processing, and speech recognition. By leveraging pre-trained models, transfer learning can reduce the amount of training data required for a new task, and can significantly speed up the training process.

We used transfer learning by initializing our CNN5 model with 2 pretrained models:

- **ResNet50** is a pre-trained CNN architecture developed by Microsoft researchers for image classification task. It consists of 50 layers, including convolutional layers, pooling layers, and fully connected layers. It uses skip connections, which enable information to bypass one or more layers in the network, and residual blocks, which allow the network to learn residual functions, to improve the accuracy of the model.

  ResNet50 is pre-trained on a large dataset of images, called ImageNet, which contains millions of images and thousands of categories. The pre-training process enables the model to learn general features of images, such as edges, textures, and shapes, which can be used for a variety of image-related tasks.

  The ResNet50 model is useful for image classification, object detection, and image segmentation tasks. It has been shown to achieve state-of-the-art performance on several benchmark datasets, and is widely used in computer vision applications.

  We built a new model by removing the top layer of the ResNet50 and a new classification layer is added on top of it to suit our specific classification problem. The weights of the ResNet50 model are frozen, so only the newly added layers are trained during the training process.

  The model is trained using data augmentation techniques to increase the diversity of the training set and improve the model's generalization capabilities. Early stopping is also used to prevent overfitting and improve the model's performance.

After training the model for 10 epochs, the model achieved an accuracy of 0.7282 on the test set. The test loss is 0.633, which indicates that the model is performing well on the classification task.

- **INCEPTIONV3 i**s a pre-trained neural network model for image classification tasks, developed by Google researchers. It's a deep learning model that can recognize patterns and objects in images with high accuracy. The name "Inception" comes from the movie of the same name, which features a dream within a dream concept, as the model includes multiple layers of nested convolutional networks.

  The InceptionV3 model is trained on a large dataset of images and can classify objects into thousands of categories. It's useful for tasks such as object detection, image recognition, and image search.

  We built a model by adding new classification layers on top layer of the INCEPTIONV3 model, froze the weights and only the newly added layers were trained during the initial training phase. The initial training was done for 10 epochs using the ***adam*** optimizer and ***early stopping*** with patience of 10 to prevent overfitting.

  After the initial training phase, we fine-tuned the model by unfreezing the last 10 layers of the model and training the entire model with a smaller learning rate. This step helps the model to learn more specific features that are relevant to the new task. We used the Adam optimizer with a learning rate of 0.001 and applied a learning rate scheduler and early stopping with patience of 3 and 10, respectively. The model was fine-tuned for 30 epochs.

The results were as follows:

	Train acc.	Train Loss	Test Acc.	Test Loss
RESNET50	0.7237	0.6375	0.7282	0.6331
INCEPTIONV3	0.8180	0.4545	0.8040	0.4293

The model using **INCEPTIONV3** achieved a **higher accuracy score** on the test dataset, which indicates that it can effectively classify new images in our specific dataset.

Exercise 2: News Classification

PROJECT BACKGROUND

The objective of this exercise is to analyze the “Reuters Newswire” dataset using Recurrent Neural Networks (RNN) and Convolutional Neural Networks (CNN). The task associated with this dataset is a multi-class classification problem, where the goal is to predict the category or categories of a given news article. This is a many-to-one type of task, where the input sequence (the text of the news article) is mapped to a single output (the predicted category or categories).

The Reuters Newswire Classification Dataset is a collection of short news articles that have been tagged with one or more categories, such as "earnings", "acquisitions", or "corn". There are 46 possible categories. The value of the target variable for each article is a binary vector of length 46, with a value of 1 indicating that the article belongs to that category and a value of 0 indicating that it does not. Therefore, the target variable for each article in the dataset is a multi-label variable. The dataset consists of 11,228 newswire articles, which have been split into a training set of 8,982 articles and a test set of 2,246 articles.

RECURRENT NEURAL NETWORKS (RNN)

RNNs are well-suited for this type of task because they are designed to handle sequential data, such as text. By processing the input sequence one element at a time, RNNs can capture the temporal dependencies that exist within the data. In the case of the Reuters newswire dataset, the order of the words in the news articles is important and can be used to help predict the category of the article.

Furthermore, RNNs are able to handle variable-length sequences, which is important for this dataset because the length of the news articles varies. By using techniques such as padding, we can ensure that all sequences have the same length, which allows us to apply RNNs to this problem.

Overall, the Reuters newswire dataset is an appropriate use case for RNNs because it involves processing sequential data of variable length, and RNNs are well-suited to handle these types of tasks. The many-to-one architecture is also appropriate for this multi-class classification problem, where the goal is to predict a single output (the category or categories of the news article) from a variable-length input sequence (the text of the news article).

Apart from simple RNN, we used GRU (Gated Recurrent Unit) and LSTM (Long Short-Term Memory) two popular types of RNNs commonly used for processing sequential data such as text,

speech, and time series data. Both models are designed to address the vanishing gradient problem that occurs when training traditional RNNs on long sequences.

In a traditional RNN, the hidden state at each time step is updated using a fixed set of weights, which can cause the gradient to vanish as the network tries to learn long-term dependencies. GRU and LSTM were designed to overcome this issue by introducing gating mechanisms that selectively allow information to flow through the network and into the hidden state.

GRU is a simpler variant of LSTM that uses two gates: a reset gate and an update gate. The reset gate controls how much of the previous hidden state should be forgotten, while the update gate controls how much of the new input should be added to the current hidden state.

LSTM, on the other hand, uses three gates: an input gate, a forget gate, and an output gate. The input gate controls how much of the new input should be added to the current hidden state, the forget gate controls how much of the previous hidden state should be forgotten, and the output gate controls how much of the hidden state should be output to the next layer.

A. Model Architecture

- RNN1: This model uses a SimpleRNN layer with 32 neurons and a dense output layer with softmax activation. It achieves a training accuracy of 0.8846 and a validation accuracy of 0.4641.
- RNN2: This model uses a GRU layer with 256 neurons, dropout of 0.9, and a GlobalMaxPooling1D layer followed by a dense output layer with softmax activation. It achieves a training accuracy of 0.7950 and a validation accuracy of 0.7435.
- RNN3: This model uses the same architecture as rnn2, but with a different optimizer (rmsprop instead of adam). It achieves a training accuracy of 0.7570 and a validation accuracy of 0.7156.
- RNN4: This model uses two GRU layers with 256 neurons each, dropout of 0.9, and a dense output layer with softmax activation. It achieves a training accuracy of 0.7886 and a validation accuracy of 0.7195.
- RNN5: This model uses two LSTM layers, one with 128 neurons and the other with 64 neurons, dropout of 0.2, recurrent dropout of 0.2, and a dense output layer with softmax activation. It achieves a training accuracy of 0.6747 and a validation accuracy of 0.5993

The results were as follows:

	Train acc.	Train Loss	Test Acc.	Test Loss
RNN1	0.8846	0.4745	0.4641	2.5826
RNN2	0.7950	0.7936	0.7435	1.1297
RNN3	0.7570	0.9647	0.7156	1.2014
RNN4	0.7886	0.7693	0.7195	1.1887
RNN5	0.6747	1.3072	0.5993	1.6588

Among these models, RNN2 has the highest validation accuracy, suggesting that it is the best performing model among the five. However, we should note that we might need to improve the model further, as the validation accuracy is still not very high, indicating that the model might be overfitting the training data. We can also see that as the model complexity increases (i.e., from rnn1 to rnn4), the training accuracy generally increases, but the validation accuracy does not necessarily follow the same trend. This suggests that the more complex models might be overfitting the training data and not generalizing well to unseen data. Finally, we can see that the test accuracy is lower than the training and validation accuracy for all the models, which is expected since the models have not seen the test data during training and might not generalize well to it.

Now we will apply attention and memory to the best performing model (RNN2) to see if the accuracy improves.

B. Attention Mechanism

The attention mechanism is a technique used in neural networks to selectively focus on certain parts of the input sequence while processing it. In traditional recurrent neural networks such as LSTM or GRU, the output at each time step depends on the entire input sequence seen so far. However, in many real-world problems, certain parts of the input sequence may be more relevant to the task at hand than others. Attention mechanisms allow the model to focus on these important parts of the input while ignoring the less important ones.

We use a Long Short-Term Memory (LSTM) RNN as the base model and add an attention mechanism on top of it. The attention mechanism learns to weigh the importance of each input element and provide a context vector that captures the most relevant information in the input sequence. Finally, the context vector is passed through a dense layer, and the output is fed into a softmax activation function to obtain a probability distribution over the 46 classes.

The results were as follows:

	Train acc.	Train Loss	Test Acc.	Test Loss
RNN6	0.9035	0.3769	0.6711	1.6341
RNN7	0.8072	0.7113	0.7295	1.1676
RNN8	0.9223	0.2890	0.6689	1.7252

Comparing the results of RNN2, RNN6, RNN7, and RNN8, we can see that RNN2 achieved the highest accuracy on the validation set with an accuracy of 0.7435, followed by RNN7 with an accuracy of 0.7295, RNN6 with an accuracy of 0.6711.Also, RNN2 had the lowest loss on the validation set of 1.1297, followed by RNN7 with a validation loss of 1.1676, RNN6 with a validation loss of 1.6341, and RNN8 with a validation loss of 1.7252.

Since RNN2 had the best performance, we can conclude that the addition of an attention mechanism did not provide much improvement in this case. However, this does not necessarily mean that attention mechanisms cannot improve the performance of the model, and it may be worthwhile to explore other attention mechanisms.

CONVOLUTIONAL NEURAL NETWORKS (CNN)

A. Model Architecture

We used 5 CNN model architectures:

- CNN1: This model uses three convolutional layers, followed by two dense layers with a dropout layer in between.
- CNN2: This model uses two convolutional layers, followed by a dense layer with a dropout layer in between. It also uses batch normalization between the convolutional and dense layers.
- CNN3: This model uses four convolutional layers, followed by three dense layers with two dropout layers in between.
- CNN4: This model uses three convolutional layers, followed by one dense layer with a dropout layer in between. It also uses L2 regularization on the dense layer.
- CNN5: This model uses two convolutional layers with L2 regularization, followed by a dense layer with a dropout layer in between.
- CNN6: This model uses two convolutional layers, followed by one dense layer with a dropout layer in between. It also uses batch normalization between the convolutional and dense layers.

B. Comparison

Although all models have achieved an accuracy above 65%, there is still room for improvement and CNN2 has the highest validation accuracy. Some possible techniques to try are adjusting the hyperparameters (such as learning rate, batch size, and number of epochs), using different activation functions or regularization methods, or trying a different model architecture altogether, such as a recurrent neural network (RNN) or transformer-based models.

The results were as follows:

	Train acc.	Train Loss	Test Acc.	Test Loss
CNN1	0.9442	0.2012	0.6555	2.7567
CNN2	0.9500	0.1805	0.7006	2.0382
CNN3	0.8792	0.4065	0.6589	2.3366
CNN4	0.9331	0.2643	0.6700	2.5783
CNN5	0.9079	0.4645	0.6784	1.5751
CNN6	0.8441	0.6226	0.6728	1.6034

The main difference between the Convolutional Neural Network (CNN) architecture and the Recurrent Neural Network (RNN) architecture is their ability to handle different types of data and their temporal dependencies.

CNNs typically use convolutional layers followed by pooling layers to extract features from the input data. Convolutional layers apply a set of filters to the input data, which helps to capture spatial patterns, while pooling layers down sample the output of the convolutional layers, which helps to reduce the computational cost of the network.

On the other hand, RNNs typically use a set of recurrent layers to process the input sequence. Recurrent layers maintain a hidden state, which is updated at each time step based on the current input and the previous hidden state. This allows the network to capture the temporal dependencies between the elements in the input sequence.

One challenge with RNNs is the vanishing gradient problem, which occurs when the gradients used to update the weights in the network become too small to be effective. This can limit the effectiveness of the network and make it difficult to train. To address this problem, several variants of RNNs have been developed, including Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), which use special gating mechanisms to control the flow of information and prevent the vanishing gradient problem.

REFRENCES:

https://towardsdatascience.com/cnn-architectures-a-deep-dive-a99441d18049
https://keras.io/api/applications/
https://www.kaggle.com/code/darthmanav/resnet-50-vgg-16-inception-v3-beginner-s-guide
https://github.com/slundberg/shap
https://www.analyticsvidhya.com/blog/2019/11/shapley-value-machine-learning-interpretability-game-theory/
https://towardsdatascience.com/deep-learning-model-interpretation-using-shap-a21786e91d16
And ChatGPT 😊