**Social Security Image Classification**

**using CNNs**

**Introduction:**

In this project, we developed a Convolutional Neural Network (CNN) to distinguish between legitimate and unrelated social security images. The goal of this study was to create an automated approach for identifying false social security images in-order to prevent fraud and identity theft.

CNNs, also referred to as convolutional neural networks, are a popular deep learning method for categorizing images. In the context of social security images, CNNs can be used to automatically categorize and recognize different kinds of documents including licenses, passports, and ID cards.

We used PyTorch, a well-liked deep learning toolbox, to build our CNN model. A set of social security card images that had been deemed relevant or acceptable served as the basis for our model's development.

**Objective:**

The objective of this study is to develop a CNN-based model as a replacement to traditional methods of social security card image verification. The current traditional approach takes a lot of time and is prone to mistakes. However, improvements in deep learning and computer vision have made it possible for us to automatically classify images of social security cards. Our study on Social Security Image Classification Using CNN seeks to create a highly accurate model capable of categorizing a variety of social security images, such as identification documents, passports, and certificates.

**Dataset:**

I used a dataset with a variety of real and distinctive documents, including passports, bills, and others, to train and test our model. Training and testing subsets of the dataset were created. About 80% of the photos were assigned to the training group, with the remaining 20% assigned to the testing subset.

To make the dataset loading and pre-processing easier, we created the CreateDataset() class. The photos were scaled to a consistent size of 200x200 pixels and transformed to a 3-channel image format as part of the pre-processing stage.

CNNs are a subset of deep learning algorithms that may be trained to recognize patterns in images. Using CNNs, social security photos can be categorized into groups like fake, real, and suspicious. To recognize image patterns and correctly classify them, these networks are trained on a sizable collection of social security image examples. To extract picture features that are important to the classification process, CNNs' convolutional layers use filters. These filters are taught to recognize specific patterns in images, which are then input into a pooling layer to reduce dimensionality. Finally, the features are processed through a fully connected layer to classify the photos. During training, CNNs combine supervised and unsupervised learning strategies using labelled and unlabelled data, respectively.

**Benefits of using CNNs:**

* High accuracy: CNNs have been capable of successfully completing image classification jobs with high accuracy. They can accurately and efficiently classify photos based on their attributes.
* Efficient: CNNs are efficient in terms of computing power and memory utilization. They are able to quickly and effectively process enormous amounts of data.
* Transfer learning: Transfer learning is the process of training CNNs on huge datasets and then applying them to different tasks. This enables the effective use of previously learned models for comparable tasks.

**Disadvantages of using CNN:**

* Complex architecture: It might be difficult to fully understand and train CNNs due to their complex construction. The network's layout and the choice of parameters can have a big impact on how accurate the model is.
* Sensitive to noise: CNNs might classify data incorrectly because they are sensitive to noise in the input. The performance of the model can be improved by pre-processing the data to remove noise.
* Limited to specific types of data: Because CNNs are made for processing images, they might not work well with other forms of data. It's crucial to choose the right algorithm for the challenge at hand.
* Limited to specific jobs: CNNs may not be appropriate for other tasks but are effective for certain functions, such as image classification. Finding the right use case and exploring alternate algorithms for other jobs is crucial for CNNs.

To effectively overcome the difficulties of employing CNNs, the following procedures can be taken:

* Utilize the most of pre-trained models: Using pre-trained models helps streamline the training process and simplify the CNN architecture.
* Use data augmentation: Data augmentation can be used to increase the diversity of the training data while decreasing sensitivity to noise.
* Apply transfer learning: A CNN may be trained on a huge dataset using transfer learning, which can then be applied to different tasks.

**Steps Involved:**

1. Loading Data: Data loading is the process of importing and getting the data ready for the model.
2. Evaluation: Analysing the model's performance on a different test set to make sure it is accurate.
3. Data Pre-processing: Preparing the data for feature extraction by cleaning, converting, and normalizing it.
4. Feature Extraction: As the name suggests, utilizing CNNs to extract key features from the data is known as feature extraction.
5. Evaluation: Analysing the model's performance on a different test set to make sure it is accurate.
6. Model training: Using a combination of supervised and unsupervised learning strategies, training the CNN model on the pre-processed data.

**Model:**

In order to use images from a given folder to train our CNN, we first created a class called CreateDataset that reads in the images, resizes them, and turns them into a usable format.

I chose ResNet-18 as the Convolutional neural network ResNet-18 has demonstrated outstanding performance on image classification challenges. It took nearly 45 mins to train the model.

With this unique architecture, three-class image recognition jobs with efficient feature extraction and classification are possible. The network can benefit from the learnt features from the substantial ImageNet dataset by using the pre-trained ResNet-18 model, potentially enhancing its performance on the particular classification job.

Steps:

1. Import all the necessary libraries required for the model.
2. We will set the path to the data folders.
3. We create a class called CreateDataset using ‘class CreateDataset: ’ .
4. In the CreateDataset class we define create\_dataset that contains (self, Train\_folder, IMG\_WIDTH, IMG\_HEIGHT) .
5. Then we read images, resize them if needed and convert them into usable format.
6. Next, we write code to display a random set of 9 images from the training set. This is the output of we got.

A close-up of a driver's license

Description automatically generated with medium confidence

1. Using PyTorch, a unique neural network architecture known as CNNNet was created. It builds on the ResNet-18 model and extends the nn.Module class.
2. Here ResNet-18 model is loaded with pre-trained weights in the \_\_init\_\_ method.
3. A new linear layer replaces the last fully connected layer of the ResNet-18 model, adapting the network for a particular classification task with three output classes.
4. The forward technique utilizes a modified ResNet-18 model to process an input tensor. The predicted class probabilities for each input are represented by the output tensor that is returned.
5. In the next step we build a method called TrainModel, which accepts our CNN model, the root directory's location, as well as our training and testing data.
6. Here our loss criterion is the nn.CrossEntropyLoss function, and our optimizer is the optim.SGD with a momentum of 0.9 and a learning rate of 0.001. Then, after iterating through a predetermined number of epochs and batches of training data, we update our weights by computing the loss and backpropagating the errors.
7. Now, we build PyTorch datasets and data loaders for CNN, load our training and test data. Then we print the output to check if the training and testing data are loaded successfully.

A screenshot of a computer

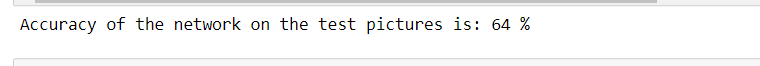
Description automatically generated with low confidence

1. Next we initialize the CNNNet model using ‘model=CNNNet()’ command.
2. Then train our CNN model using the TrainModel method. After that, the trained model is stored in the designated directory. Then we print the output to see if the training is complete.

A screenshot of a computer

Description automatically generated with low confidence

1. In the next step, we evaluate the model using ‘model.eval()’ and then calculate and print the accuracy of the network on the test pictures.



1. In the final step, we assume our input size and then print the model summary.

**Results:**

On the test set, our CNN model had an accuracy of 64%. This shows that the model was only partially successful in identifying social security images from other documents, ID’s, as well as from other irrelevant papers.

**Conclusion**:

Finally, we have successfully created a CNN model that can identify real and irrelevant social security images. Our model performed well but not as expected on the test set, implying that it could be further developed into a useful tool for detecting falsify social security images.

**Summary of the model:**

A screenshot of a computer

Description automatically generated with medium confidence

A screenshot of a computer

Description automatically generated