T.C.

AKDENİZ ÜNİVERSİTESİ

GÖRÜNTÜLERDEKİ EKSİK KISMIN TAMAMLANMASI AMAÇLI DERİN ÖĞRENME TEMELLİ HİBRİT BİR MODEL GELİŞTİRİLMESİ

Hasan Basri AKÇAY

FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

HAZİRAN 2022

ANTALYA

T.C.

AKDENİZ ÜNİVERSİTESİ

GÖRÜNTÜLERDEKİ EKSİK KISMIN TAMAMLANMASI AMAÇLI DERİN ÖĞRENME TEMELLİ HİBRİT BİR MODEL GELİŞTİRİLMESİ

Hasan Basri AKÇAY

FEN BİLİMLERİ ENSTİTÜSÜ

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI

YÜKSEK LİSANS TEZİ

HAZİRAN 2022

ANTALYA

T.C.
AKDENİZ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GÖRÜNTÜLERDEKİ EKSİK KISMIN TAMAMLANMASI AMAÇLI DERİN ÖĞRENME TEMELLİ HİBRİT BİR MODEL GELİŞTİRİLMESİ

Hasan Basri AKÇAY

ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ

ANABİLİM DALI

YÜKSEK LİSANS TEZİ

Bu tez 28/06/2022 tarihinde jüri tarafından Oybirliği ile kabul edilmiştir.

Doç. Dr. Övünç POLAT (Danışman)
Prof. Dr. Ömer Halil ÇOLAK
Dr. Öğr. Üyesi Özge ÖZTİMUR KARADAĞ

ÖZET

GÖRÜNTÜLERDEKİ EKSİK KISMIN TAMAMLANMASI AMAÇLI DERİN ÖĞRENME TEMELLİ HİBRİT BİR MODEL GELİŞTİRİLMESİ

Hasan Basri AKÇAY

Yüksek Lisans Tezi, ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ

Danışman: Doç. Dr. Övünç POLAT

Haziran 2022; 30 sayfa

Bu tez çalışmasında görüntülerin eksik kısımlarının tahminlenmesi üzerine Konvolüsyonel Sinir Ağı (CNN), Uzun Kısa Süreli Bellek (LSTM) temelli hibrit bir model ve Otokodlayıcı (Autoencoder) metodu tasarlanmıştır. Görüntülerdeki eksik kısımların tahmini, gönderim işlemi yarıda kesilmiş, sinema moduna uymayan ve istenmeyen bir nesneye sahip olan görüntüler için oldukça önemlidir. Çalışma Cifar-10 veri seti üzerinde gerçekleştirilmiştir. Otokodlayıcı görüntülerin anlamlı bir şekilde sıkıştırılmasında kullanılmıştır. Sıkıştırılan görüntülerin veri dağılımlarının kontrolü için Üretken Karşıt Ağlar (GANs) metodu kullanılmış ve sıkıştırılan görüntülerin dağılımı normal dağılıma yakınsanmıştır. CNN modelinin girdileri Cifar-10 görüntüleriyken LSTM modelinin girdileri Otokodlayıcı modelinin çıktısı olan sıkıştırılmış görüntü verileridir.

Özellik çıkarma işlemi hem CNN hem LSTM modeli tarafından gerçekleştirilmiş ve çıkarılan özellikler birleştirilmiştir. Bu özellikler daha sonra yeni bir LSTM ve CNN modeline girdi olarak verilerek görüntülerin eksik kısımları tahmin edilmiştir. CNN ve LSTM iki farklı tahmin üretmektedir. Modelin eğitimleri sırasında hata metriği olarak Ortalama Karekök Sapması (RMSE), Yapısal Benzerlik İndeks Hesaplaması (SSIM), Üretken Karşıt Ağlar Kaybı ve VGG (Visual Geometry Group, önceden eğitilmiş model) kaybı kullanılmıştır. VGG kaybı daha önceden eğitilmiş VGG modelinin çıktıları arasındaki farkı ifade eder. Geliştirilen modelin sonuçları klasik yöntemler ile karşılaştırılmış ve tablo halinde paylaşılmıştır.

Literatürde görüntülerin eksik kısımlarının tahminlenmesi için sıklıkla görüntü işleme metotları kullanılırken bu çalışmada hibrit bir model oluşturulmuştur. Hibrit model çıktısı LSTM çıktısının ve CNN çıktısının ağırlıklı ortalamasından oluşmaktadır. Hibrit model çıktısı, test veri seti üzerinde RMSE, Sinyal Gürültü Oranı (PSNR), SSIM ve Frechet Başlangıç Mesafesi (FID) metriklerinde test edilen modeller içerisinde en yüksek puanları alarak bu problem için uygunluğunu kanıtlamıştır.

ANAHTAR KELİMELER: Görüntü Tamamlama, Konvolüsyonel Sinir Ağı, Otokodlayıcı, Uzun Kısa Süreli Bellek, Üretken Karşıt Ağlar, VGG

JÜRİ: Doç. Dr. Övünç POLAT

Prof. Dr. Ömer Halil ÇOLAK

Dr. Öğr. Üyesi Özge ÖZTİMUR KARADAĞ

ABSTRACT

DEVELOPMENT OF A DEEP LEARNING BASED HYBRID MODEL TO COMPLETE THE MISSING PART IN THE IMAGES

Hasan Basri AKÇAY

MSc / PhD Thesis in Electrical-Electronics Engineering

Supervisor: Assoc. Prof. Dr. Övünç POLAT

June 2022; 30 pages

In this thesis, a hybrid model based on Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM) and Autoencoder method were designed to predict missing parts of images. The estimation of missing parts in images is very important for images that have been interrupted, do not fit the cinema mode, and have an undesirable object. The study was worked on the Cifar-10 dataset. The autoencoder is used to compress the images in a meaningful way. Generative Adversarial Networks (GANs) method was used to control the data distributions of the compressed images and the distribution of the compressed images converged to the normal distribution. While the inputs of the CNN model are Cifar-10 images, the inputs of the LSTM model are compressed image data, which is the output of the Autoencoder model.

Feature extraction was performed by both the CNN and LSTM model, and the extracted features were combined. These features were then given as input to a new LSTM and CNN model, and the missing parts of the images were estimated. CNN and LSTM generate two different forecasts. Root Mean Square Deviation (RMSE), Structural Similarity Index Calculation (SSIM), Generative Adversarial Networks Loss, and VGG (Visual Geometry Group, pre-trained model) loss were used as error metrics during the training of the model. The VGG loss represents the difference between the outputs of the previously trained VGG model. The results of the developed model were compared with the classical methods and shared in a table.

While image processing methods are frequently used in the literature to predict missing parts of images, a hybrid model was created in this study. The hybrid model output consists of the weighted average of the LSTM output and the CNN output. The hybrid model output proved its suitability for this problem by getting the highest scores among the models tested on the RMSE, Signal-Noise Ratio (PSNR), SSIM and Fréchet Initial Distance (FID) metrics on the test dataset.

KEYWORDS: Autoencoders, CNN, GANs, Image Completion, LSTM, VGG

COMMITTEE: Assoc. Prof. Dr. Övünç POLAT

Prof. Dr. Ömer Halil ÇOLAK

Asst. Prof. Dr. Özge ÖZTİMUR KARADAĞ

ÖNSÖZ

Dijitalleşme günümüzde hız kesmeden devam etmektedir. Dijitalleşmeyle beraber ekranlar daha büyük hale gelmekte ve hatta sanal ortamlar oluşturulmaktadır. Görüntü kalitesi bu geniş ekranlar ve sanal ortamlar için çok önemlidir fakat tüm görüntüler yüksek çözünürlüklü değildir. Yüksek çözünürlüklü görüntüler de çok fazla hafıza kapladığı için çok sık tercih edilmemektedir. Bu çalışma kapsamında görüntülerin devamının gerçek zamanlı tahminleri yapılarak görüntü çözünürlüğünün arttırılması hedeflenmiştir.

Tez çalışmamın yürütülmesi esnasında yardım ve desteklerini benden esirgemeyen, bilgisinden ve tecrübelerinden faydalandığım, danışmanım Doç. Dr. Övünç POLAT'a (Akdeniz Üniversitesi Mühendislik Fakültesi) en içten teşekkürlerimi sunarım.

İÇİNDEKİLER

AKADEMİK BEYAN

Yüksek Lisans Tezi olarak sunduğum "Görüntülerdeki Eksik Kısmın Tamamlanması Amaçlı Derin Öğrenme Temelli Hibrit Bir Model Geliştirilmesi" adlı bu çalışmanın, akademik kurallar ve etik değerlere uygun olarak yazıldığını belirtir, bu tez çalışmasında bana ait olmayan tüm bilgilerin kaynağını gösterdiğimi beyan ederim.

28/06/2022

Hasan Basri AKÇAY

KISALTMALAR

Kısaltmalar

CNN : Convolutional Neural Network (Konvolüsyonel Sinir Ağı)

GANs : Generative Adversarial Networks (Üretken Karşıt Ağlar)

LSTM : Long Short-Term Memory (Uzun Kısa Süreli Bellek)

RMSE : Root Mean Square Error (Ortalama Karekök Sapması)

SSIM : Structural Similarity Index Measure (Yapısal Benzerlik İndeks Hesaplaması)

VGG : Visual Geometry Group (Önceden eğitilmiş bir model)

YSA : Yapay Sinir Ağları

PCA : Principal Component Analysis (Temel Bileşenler Analizi)

PSNR : Peak Signal-to-Noise Ratio (En Yüksek Sinyal-Gürültü Oranı)

FID : Frechet Inception Distance (Frechet Başlangıç Mesafesi)

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

Tarih boyunca insanlar yapmış oldukları eylemleri kayıt altına alma eğiliminde olmuşlardır. Eski çağlarda bu durum kendini mağara duvarına resim çizmek ile gösterirken yeni çağlarda ise çivi yazısı olarak göstermiştir. Yazının icadı dünya insanlık tarihindeki devrimlerden ilkidir ve insanlığın macerasını değiştirecek bilgilerin saklanması, doğru bir şekilde kaydedilmesi için bir araç olmuştur. Bu yüzden en büyük devrimlerden biri olmayı hak etmiştir (Hakan Kutluay vd 2016). Zaman ilerledikçe kayıt sistemlerimiz hem daha kaliteli hem de daha küçük bir hale gelmiştir. Kayıt sistemlerinin bu kadar gelişmesinin sebebi dijitalleşmedir. Delikli kart ile başlayan modern veri kaydı dijitalleşme ile bulut veri tabanlarına kadar taşınmıştır. Bulut teknolojileri, sanal ortamda barındırılan uygulamalara veya verilere internet üzerinden her yerden ulaşım imkanı sağlayarak, uzun bir zaman diliminde gerçekleşen veri işlemlerinin diğer yöntemlerle en kısa sürede yapılmasını basitleştirmiştir (Ataç & Akleylek vd 2019).

Günümüzde dijitalleşme hız kesmeden devam etmektedir ve kayıt altına alınan en önemli dosyalardan birkaçı görüntü ve video dosyalarıdır. Görüntü ve video dosyaları Geçmiş dönemi görmemizi sağlayan bir nevi zaman makineleridir. Bu dosyalar sayesinde şehirlerin eski haline, şehirlerin eski bitki örtüsünü hatta nesli tükenmiş bazı hayvanları dahi görmek mümkündür. Bu durum fotoğrafçılığın ne kadar önemli olduğunu bizlere göstermektedir. 1839 yılında başlayan fotoğraf sözcüğü, Yunancaya uygun olarak "vossıshe Zeitung" dergisinde "Photographie", ışık "fotos" ve yazı "graphe" kelimelerinin birleşmesinden oluşmuştur (Ayşe Kahraman vd 2020). Fotoğraflar günümüzde genellikle dijital görüntü olarak kaydedilir ve dijital görüntüler, piksel olarak da bilinen resim öğelerinden oluşmaktadır. Piksel, sırasıyla x ekseni ve y ekseninde belirtilen uzamsal koordinatlarla girdi olarak beslenen iki boyutlu fonksiyonların çıktısı olan gri alan veya her bir yoğunluk için sonlu, ayrık miktarların sayısal temsilini içeren bir görüntüdür (Gonzalez, Rafael vd 2018).

Çevrimiçi olarak yakalanan ve paylaşılan dijital görüntülerin sayısı olağanüstü bir oranda artmaktadır. Yakın zamanda, Facebook'un şu anda 15 milyar görüntü (replikasyonlar hariç) depoladığı ve her ay 850 milyon yeni fotoğraf eklendiği bildirildi. Sadece Facebook değil, aynı zamanda bugüne kadarki en büyük fotoğraf paylaşım hizmeti olan ImageShack, 20 milyar görüntüye ev sahipliği yaparken, News Corp'un PhotoBucket ve Yahoo'nun Flickr'ı sırasıyla 7,2 ve 3,4 milyar fotoğraf depolamaktadır (Sivic, J., Kaneva, B., Torralba, A., Avidan, S. vd 2008).

Birçok farklı disiplinde, yüksek kaliteli görüntü uzantılarına güçlü bir ihtiyaç vardır. Örneğin sanal gerçeklikte, genellikle bir görüntüyü yakalamak için kullanılandan farklı kamera dış özelliklerini simüle etmek gerekir ve bu genellikle içeriğin orijinal görüntü sınırlarının dışında doldurulmasını gerektirir (Krishnan, D., Teterwak, P., Sarna, A., Maschinot, A., Liu, C., Belanger, D. vd 2019). Bunların dışında kaliteli ve geniş görüntülerin izleyiciler için daha sürükleyici deneyimler yarattığı görülmüştür (Agarwala, A., Zheng, K. C., Pal, C., Agrawala, M., Cohen, M., Curless, B., Salesin, D., & Szeliski, R vd 2005).

Bu çalışmada orijinal görüntüler ile genişletilmiş görüntüler arasında resim yapısını bozmayan bir uyumun yakalanması hedeflenmektedir. Bunun dışında

genişletilmiş bölge, yapısal, semantik ve dokusal düzeylerde orijinal ile uyumlu olmalı, makul bir uzantı olarak görünmelidir. Uzantı tahmini, genişletilecek görüntünün yalnızca bir tarafına uygulanmıştır. Bu problem, doldurulacak bölgenin orijinal görüntü verileriyle her yönden eşleştiği ve sorunu önemli ölçüde kısıtladığı görüntü içi boyama problemidir. Bu nedenle, iç boyama algoritmaları, görüntü uzatma algoritmalarından daha öngörülebilir ve daha kaliteli sonuçlara sahip olma eğilimindedir. Hiçbir değişiklik yapılmadan iç boyama algoritmalarının kullanılması, görüntü uzantısı tahminlerinde kötü sonuçlara yol açmaktadır (Krishnan, D., Teterwak, P., Sarna, A., Maschinot, A., Liu, C., Belanger, D. vd 2019).

2. KAYNAK TARAMASI

Liu, G., Reda, F. A., Shih, K. J., Wang, T.-C., Tao, A., & Catanzaro, B. (2018), görüntülerdeki eksik kısımları tamamlama amaçlı olarak PConv tekniğini geliştirmişlerdir. PConv, sadece görüntüde bulanan pikseller üzerinde filtrelemeler yapıldığı, görüntüde bulunan kayıp piksellerin işleme dahil edilmediği bir modeldir. Başarı kriteri olarak kendi ürettiği resimleri ve başka inpainting yöntemleri tarafından üretilmiş aynı resimleri bir ankette toplayarak gerçek insanlara hangi resmin daha gerçekçi gözüktüğünü sormuşlardır.

Steven Cheng-Xian Li, Bo Jiang ve Benjamin M. Marlin. (2019), görüntülerdeki eksik kısımları tamamlama amaçlı olarak MISGAN modelini geliştirmişlerdir. MISGAN yöntemi ile Mnist veri seti üzerinde çalışmışlardır. MISGAN modelinin iki adet çıktısı bulunmaktadır. Bu çıktılardan ilki eksik kısımları tamamlanmış görüntüyken diğeri sadece görüntünün eksik kısmı üzerine yapılmış olan tahmindir. Başarı yöntemi olarak kendi oluşturduğu CNN ve FC MisGAN modellerini bağımsız kayıplar ve dörtgen kayıplar üzerinde test etmişlerdir.

Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan ve Ilya Sutskever. (2020), görüntülerdeki eksik kısımları tamamlama amaçlı olarak Bert Transformer tekniğinden yararlanmışlardır. Bert 2018 yılında Google tarafından üretilen bir transformer modeli olup doğal dil işleme problemleri için üretilmiştir. Bert transformer tekniğinin sonuna kendi sınıflandırma için eğittiği bir modelin katmanlarını eklemiş ve bert modeli ile bu katmanın çıktısını tahmin etmeye çalışmışlardır. Katmanın çıktısını tahmin ettikten sonra bu çıktıyı sağlayacak resmi oluşturup sınıflandırma işlemine tabi tutmuşlardır. Sınıflandırma sonucu başarı kriterleridir.

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, Koray Kavukcuoglu (2016), görüntülerdeki eksik kısımları tamamlama amaçlı olarak Gated PixelCNN tekniğini geliştirmişlerdir. Gated PixelCNN filtreler yardımıyla görüntü üretirken sadece filtrenin önceden piksel oluşturulmuş kısımlarını kullanır. Gated PixelCNN tekniği ile cifar-10 veri seti üzerinde çalışmışlardır. Başarı kriterleri oluşturulan resimlerin doğru sınıflandırılmasıdır. Geleneksel convolutional auto-encoder mimarisinde deconv yerine Gated PixelCNN tekniğini uygulamışlar ve accuracy'nin yükseldiğini görmüşlerdir. Bunun yanında Gated PixelCNN gördüklerine olma ihtimali daha yüksek pixeller ekleyerek soyut görüntülerde çıkarmışlardır.

Gao, C., Saraf, A., Huang, J.-B., & Kopf, J. (2020), görüntülerdeki eksik kısımları tamamlama amaçlı olarak Flow-edge tekniğini geliştirmişlerdir. Flow-edge modeli görüntünün hareket haritası ve kenarlarını çıkarmaktadır. Daha sonrasında orijinal göründen yola çıkarak görüntü tekrar renklendirilmektedir. Bu teknik videolardaki resimlerin eksik kısımlarını tamamlayarak video bütünlüğünü bozmamayı hedefler. Başarı kriteri olarak PSNR, SSIM ve LPIPS tekniklerini kullanılmışlardır.

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016), görüntülerdeki eksik kısımları tamamlama amaçlı olarak Encoder ve Decoder ve Adversarial loss yöntemini kullanmışlardır. Otomatik kodlayıcının girdisi eksik kısımları bulunan görüntüyken çıktısı sadece görüntüde bulunan eksik kısımlardır. Mean L1 Loss, Mean L2 Loss ve PSNR teknikleri ile başarıyı ölçmüşlerdir.

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., & Huang, T. S. (2018), görüntülerdeki eksik kısımları tamamlama amaçlı olarak 2 farklı CNN modelini beraber kullanmayı önermişlerdir. Bu iki model hem eksik kısım hem de oluşturulan resmin tamamı için bir tahminde bulunur. WGAN-GP adversarial loss ve başarı kriteri olarak L1 loss, L2 loss, PSNR ve TV loss kullanmışlardır.

Kasaraneni, S. H., & Mishra, A. (2020), görüntülerdeki eksik kısımları tamamlama amaçlı olarak Cycle GAN yöntemini önermişlerdir. CycleGAN, eşleştirilmiş örnekler olmadan görüntüden görüntüye çeviri modellerinin otomatik eğitimini içeren bir tekniktir. Cycle GAN yöntemi ve farklı loss fonksiyonları beraber kullanmayı önermiş ve Cycle loss başarı kriterleridir.

Zheng, C., Cham, T.-J., & Cai, J. (2019), görüntülerdeki eksik kısımları tamamlama amaçlı olarak GANs yöntemini önermişlerdir. Başarı kriterleri L1 loss, PSNR, TV loss ve IS'dir.

Nazeri, K., Ng, E., Joseph, T., Qureshi, F., & Ebrahimi, M. (2019), görüntülerdeki eksik kısımları tamamlama amaçlı olarak Kenar üretme yöntemi ve GANs'i beraber kullanmayı önermişlerdir. Başarı kriteri olarak L1, SSIM, PSNR ve FID'i kullanmışlardır.

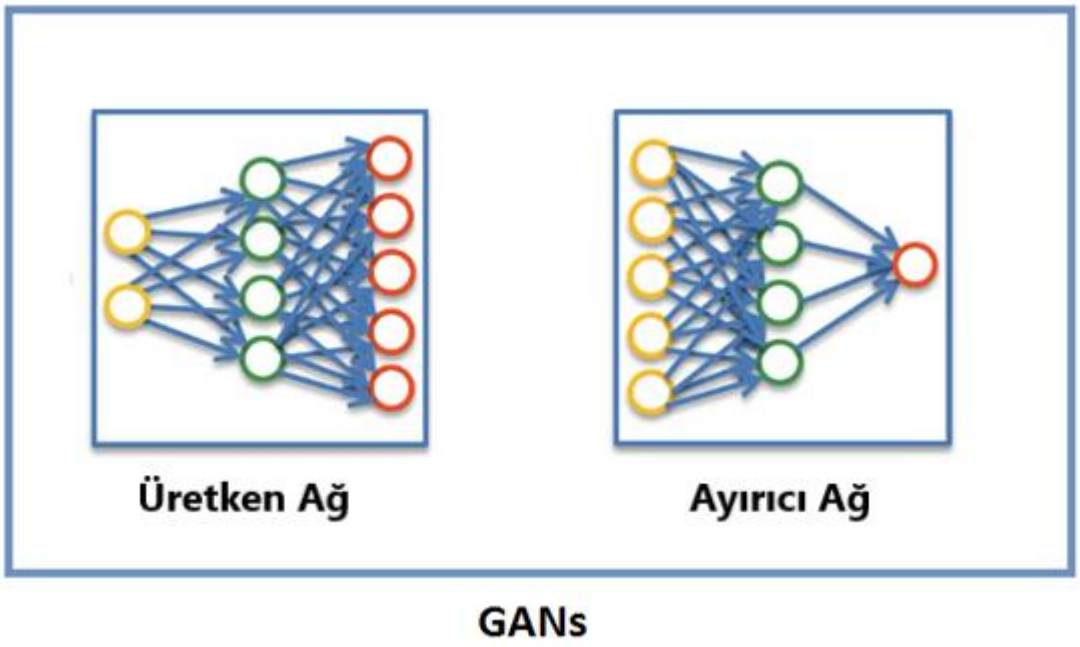
Krishnan, D., Teterwak, P., Sarna, A., Maschinot, A., Liu, C., Belanger, D., & Freeman, W. (2019), görüntülerdeki eksik kısımları tamamlama amaçlı olarak GANs yöntemini önermişlerdir. Başarı kriteri olarak belli skor metrikleri kullanmıştır. Model reconstruction loss ve adversarial loss kullanmıştır.

Literatürdeki çalışmalar incelendiğinde görüntüdeki eksik kısımları tamamlama kapsamında yapay zekâ temelli olarak bugüne kadar genelde CNN kullanılmıştır. Bununla beraber 2020 tarihli Generative Pretraining from Pixels (Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan ve Ilya Sutskever, 2020) makalesi Doğal Dil İşleme yöntemlerinin de görüntü tamamlama için kullanılabileceğini bizlere göstermiştir. Doğal Dil İşleme modeli olarak BERT modeli kullanılmıştır. Bu tez çalışmasında hem CNN hem de bahsedilen makaleden farklı olarak LSTM modellerinin bulunduğu hibrit bir model geliştirerek görüntülerdeki eksik kısımları en iyi şekilde tamamlayacak bir yöntemin geliştirilmesi hedeflenmiştir.

3. MATERYAL VE METOT

3.1. Üretken Karşıt Ağlar (GANs)

Önerilen çekişmeli ağlar çerçevesinde, üretici model bir rakiple karşılaştırılır. Bu rakip bir ayırıcı modeldir ve girdinin model dağılımından mı yoksa veri dağılımından mı olduğunu belirlemeye çalışır. Ayrımcı model, sahte parayı tespit etmeye çalışan polise benzerken üretken model, sahte para üretmeye ve tespit etmeden kullanmaya çalışan bir kalpazan ekibi gibi davranır. Bu oyundaki rekabet içerisinde iki model yöntemlerini geliştirmeye devam eder ta ki sahte ürünler gerçek ürünlerden ayırt edilemez hale gelene kadar (Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio vd 2014). Yann LeCun, bu modeli eğitim fikrinden dolayı "Makine Öğreniminde son 10 yılın en ilginç fikri" olarak nitelendirdi (Rocca, J. vd 2021).



Şekil 3.1. Üretken Karşıt Ağlar

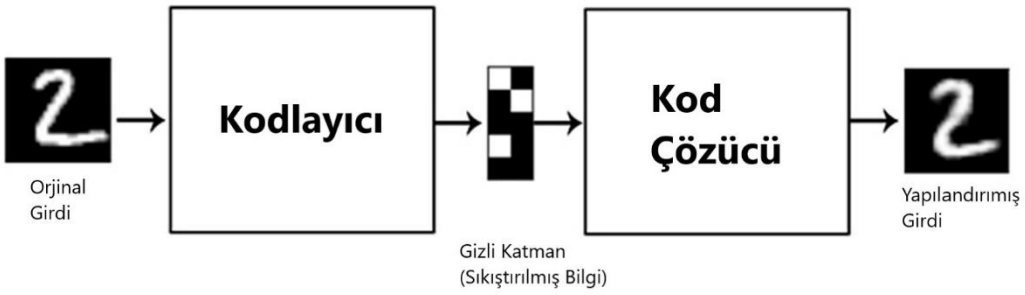
Şekil 3.1'de görüldüğü üzere bir Üretken karşıt Ağ iki farklı yapay sinir ağı modelinin birleşmesinden oluşur. Bu yapay sinir ağları Üretken Yapay Sinir Ağı ve Ayırıcı Yapay Sinir Ağı'dır.

Üretken Karşıt Ağlar bünyesinde iki farklı modeli barındırdığı için 2 farklı eğitime tabi tutulur. Eğitimlerden birinde ayırıcı model eğitilirken diğerinde üretken model eğitilir ve bu döngü devam eder. Döngünün bir adımı şu şekildedir; ilk olarak veri setindeki veriler ve üretken ağın ürettiği resimler bir veri seti olarak ayarlanır ve veri setindeki değerlere 1, üretken ağın ürettiği resimlere 0 etiketi atanır veya tam tersi de olabilir. Bu oluşturulan veri seti ile ayırıcı model eğitilir ve üretken modelin ürettiği

resimlerle gerçek resimlerin farkını anlamaya başlar. Daha sonra gürültü verileri veya üretilen resmin bağlı olması istenilen bir uzay Üretken Karşıt Ağ modeline yani hem ayırıcı hem üreticinin bulunduğu modele girdi olarak verilip sonucu gerçek veri setinin etiketi olacak şekilde eğitilir. Bu eğitim işlemi sırasında ayırıcı modelin ağırlıkları sabitlendiği için sadece üretken model eğitilir ve ayırıcı modeli yanıltacak şekilde eğitilir. Bu işlem tekrar tekrar devam eder ta ki üretken model ve ayırıcı model mükemmel sonuçlar üretene kadar.

3.2. Otokodlayıcılar

Otomatik kodlayıcılar ilk olarak 1986'da girdisini yeniden yapılandırmak için eğitilmiş bir sinir ağı olarak tanıtıldı (Rumelhart, D.E., Hinton, G.E., Williams, R.J vd 1986). Temel amaçları, kümeleme gibi çeşitli uygulamalar için kullanılabilecek verilerin "bilgilendirici" bir temsilini denetimsiz bir şekilde öğrenmektir (Dor Bank, Noam Koenigstein, Raja Giryes 2021).



Şekil 3.2. Otomatik Kodlayıcılar

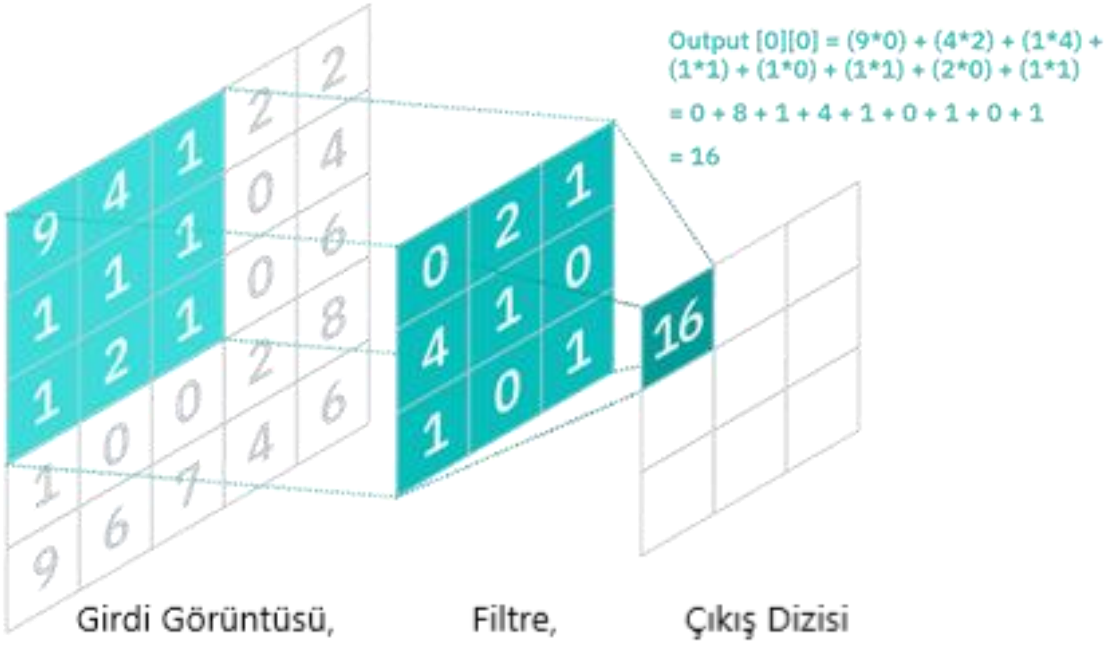
En popüler otomatik kodlayıcı modeli Şekil 3.2'de gösterilmiştir. Otomatik kodlayıcılar günümüzde veri boyutu azaltma, özellik çıkarma, gürültü giderme, görüntü renklendirme ve yeni veri üretiminde kullanılmaktadırlar (Öngün, C. vd 2020).

Otomatik kodlayıcılar iki farklı YSA'dan oluşur. İlk YSA Kodlayıcı (encoder) görüntünün sıkıştırılmasını hedefler. Örneğin 24x24 boyutundaki bir görüntünün piksel sayısı 576'iken encoder görüntüyü 100x1 boyutunda bir vektöre dönüştürür. Bu vektör ilk görüntüden daha az boyuttadır ve anlamsız gürültüler gibi gözükür. Oluşturulan bu vektör ikinci YSA Kod Çözücü'ye (decoder) girdi olarak verildiğinde ise vektör eski görüntü formatına dönüşür. En basit haliyle otomatik kodlayıcılar bu şekilde çalışmaktadırlar.

3.3. Konvolüsyonel Sinir Ağı (CNN)

Bir Konvolüsyonel Sinir Ağı (CNN), bir girdi görüntüsünü alabilen, görüntüdeki çeşitli yönlere/nesnelere ağırlık (öğrenilebilir ağırlıklar ve önyargılar) atayan ve birini diğerinden ayırt edebilen bir Derin Öğrenme algoritmasıdır (Saha, S. vd 2018).

Konvolüsyonel Sinir Ağları (CNN'ler), öğrenme yoluyla kendi kendini optimize eden nöronlardan oluştukları için geleneksel YSA'larına benzer. Her nöron bir girdidir ve skalar çarpım gibi operasyonlara tabi tutulur. Son katman, sınıflarla ilişkili kayıp fonksiyonlarını içerecektir ve geleneksel YSA'lar için geliştirilmiş tüm düzenli ipuçları ve püf noktaları hala geçerlidir. CNN'ler ve geleneksel YSA'lar arasındaki tek dikkate değer fark, CNN'lerin öncelikle görüntüler içinde örüntü tanıma alanında kullanılmasıdır (Keiron O'Shea and Ryan Nash vd 2015).



Şekil 3.3. Konvolüsyonel Sinir Ağı

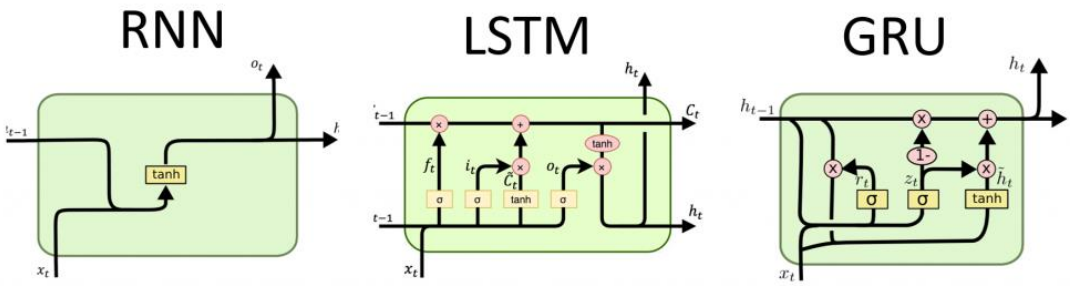
Şekil 3.3'te 5x5 büyüklüğündeki bir görüntünün 3x3 büyüklüğünde bir çekirdek ile işlemini görmektesiniz. 3x3 büyüklüğündeki çekirdek giriş görüntüsü üzerinde gezdirilir ve örtüşen sayılar çarpılıp çıkan sonuçlar toplanarak yeni katmanın değerini oluşturur. Örneğin yeni katmanın ilk değeri (9*0) + (4*2) + (1*1) + (1*4) + (1*1) + (1*0) + (1*1) + (2*0) + (1*1) işlemi ile hesaplanarak 16 değeri elde edilir.

CNN yapay zeka modellerinin, nesne tanıma, yüz tanıma, nesne takibi, özellik çıkarma, görüntü bölütleme, görüntü oluşturma, çözünürlük arttırma, derinlik algılama gibi geniş bir kullanım alanları vardır.

3.4. Uzun Kısa Süreli Bellek (LSTM)

1997'de LSTM derin öğrenme algoritması, RNN mimarisinin uzun süreli zaman serilerini hatırlayamama dezavantajını yok etmek için Hochreither ve Schmidhuber tarafından ortaya atılan bir tekrarlayan sinir ağı olarak bilinmektedir (Chakraborty, K., Mehrotra, K., Mohan, C.K., and Ranka, S., 1992). Hafıza geçişli mekanizması ile

LSTM, uzun vadeli bağlamları öğrenebilmesi sayesinde zaman serisi veya sıralı problemler için oldukça uygundur (Süzen, A. A. 2019).

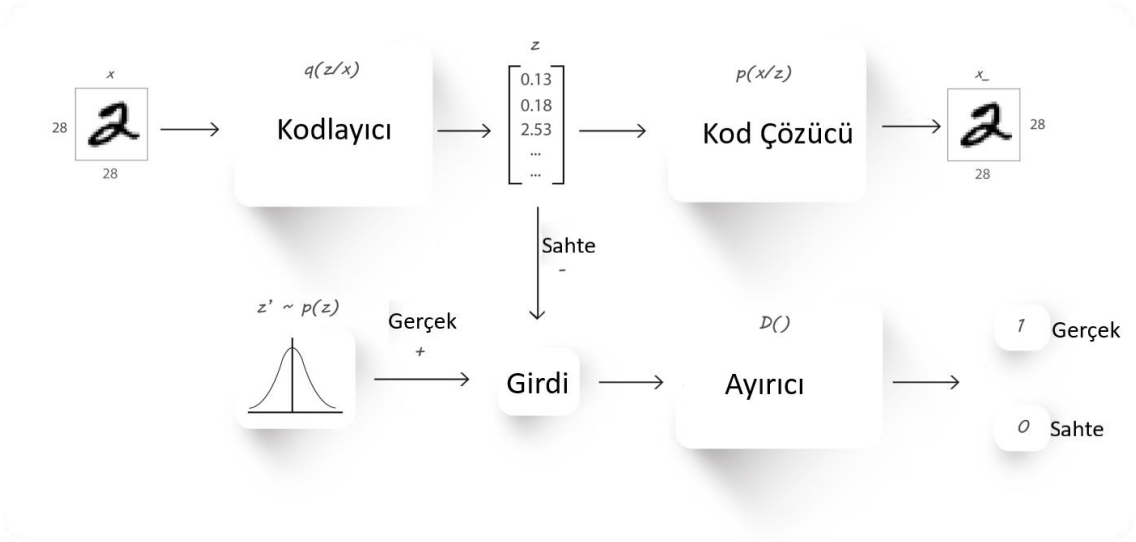


Şekil 3.4. Uzun Kısa Süreli Bellek (LSTM)

Şekil 3.4'te LSTM modelinin RNN ve GRU modellerinden farkını görmektesiniz. Bu şekle göre standart RNN modellerinin bir adet tanh katmanı bulunurken LSTM modellerinde birbirleriyle iletişim halinde bulunan dört farklı katman bulunur. Bu katmanlardan Hücre Durumu (Cell State) tahminler için anlamlı bilgileri hücreler arası taşır, bu sayede model eski verileri kısa süreli hafızasında tutmuş olur. Unutma Kapısı (Forget Gate), hatırlanması ve unutulması gereken bilgiler hakkında kararlar alan katmandır. Giriş Kapısı (Input Gate), Hücre Durumu güncellemesi gerçekleştirir. Güncelleme işleminin gerçekleşip gerçekleşmeyeceği sigmoid fonksiyonun çıktısına bağlıdır. Son olarak Çıkış Kapısı (Output Gate), kendisinden sonra gelen hücrenin girişini belirler ve tahminler için kullanılır (Onur Akköse, 2020, Aralık 22).

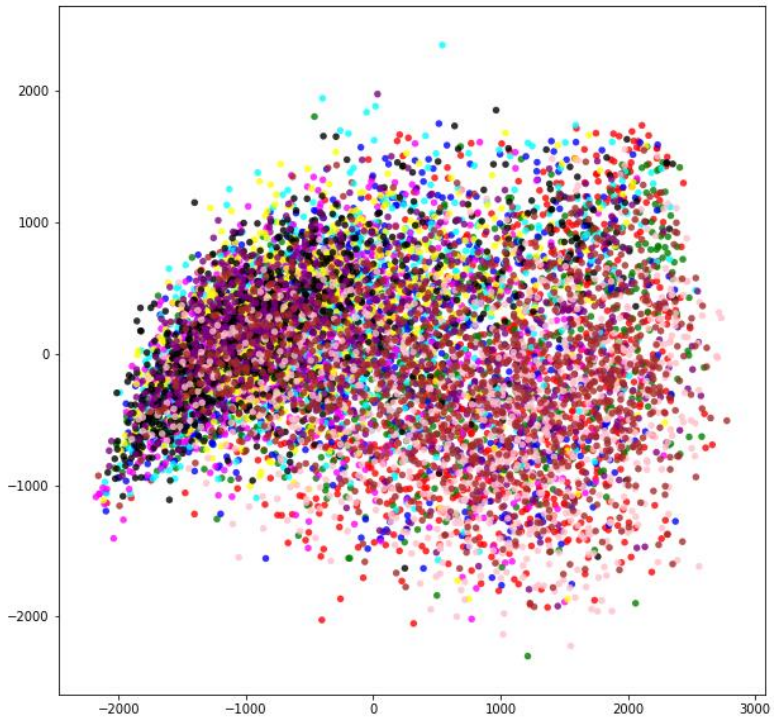
3.5. Hibrit Model

Araştırmada hem görüntü işleme hem de zaman serisi modellerinin bir arada bulunduğu hibrit bir model tasarlanmıştır. Bu modelin iki adet girdisi ve çıktısı bulunmaktadır. İlk girdi görüntünün %25'i kesilmiş resmi içerirken ikinci girdi %25'i kesilmiş görüntünün otomatik kodlayıcı tarafından sıkıştırılmış halini içerir. Otomatik kodlayıcının mimarisi aşağıda görüldüğü gibidir.

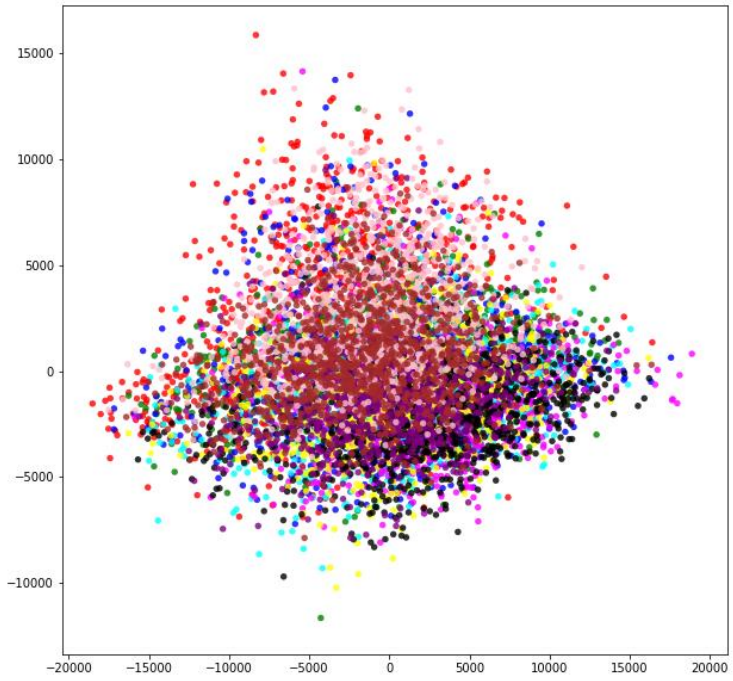


Şekil 3.5.1. Otomatik Kodlayıcı Modeli (Nagabushan, N., 2017, Kasım 28)

Şekil 3.5.1'de görmüş olduğunuz otomatik kodlayıcının gizli katmanına ayırıcı model eklenmiştir. Bu model gizli katman değerini 0, normal dağılım değerini 1 olarak tahmin etmeye çalışır. Ayrıcı modelinin kaybı rekabetçi kayıp olarak bilinmektedir. Bunun yapılmasının ana sebebi gizli katmandaki veri dağılımını normal dağılıma yakınsamaktır. Normal dağılıma sahip verilerin yapay zeka modellerinin tahminleri için daha uygun olduğu bilinmektedir.



Şekil 3.5.2 Otomatik Kodlayıcı Gizli Katman Rekabetçi Kayıpsız



Şekil 3.5.3 Otomatik Kodlayıcı Gizli Katman Rekabetçi Kayıplı

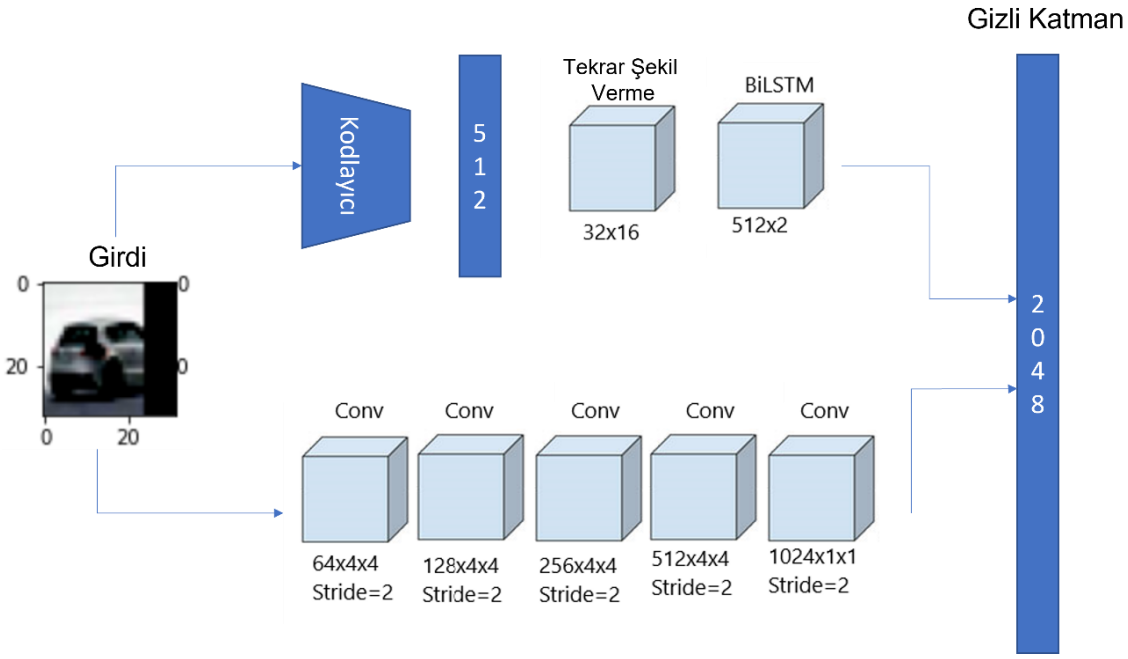
Şekil 3.5.2 ve 3.5.3'te PCA tekniği ile rekabetçi kaybının modelin gizli katmanını nasıl etkilediğini görmektesiniz. PCA tekniği ile 512'lik boyut 2'ye indirilerek görselleştirilebilir formata getirilmiştir. Şekil 3.5.2'de gizli katmanın sonuçları daha dağınıkken Şekil 3.5.3'te GANs modelinin rekabetçi kayıp fonksiyonu kullanılmasıyla beraber gizli katman sonuçları normal dağılıma daha fazla yakınsanmıştır.

Gizli katman değeri için sırası ile 64, 128, 256, 512, 1024 değerleri sırasıyla nedenmiş ve bilgi kaybı olmadan seçilebilecek minimum gizli katman boyutu 512 olarak belirlenmiştir.

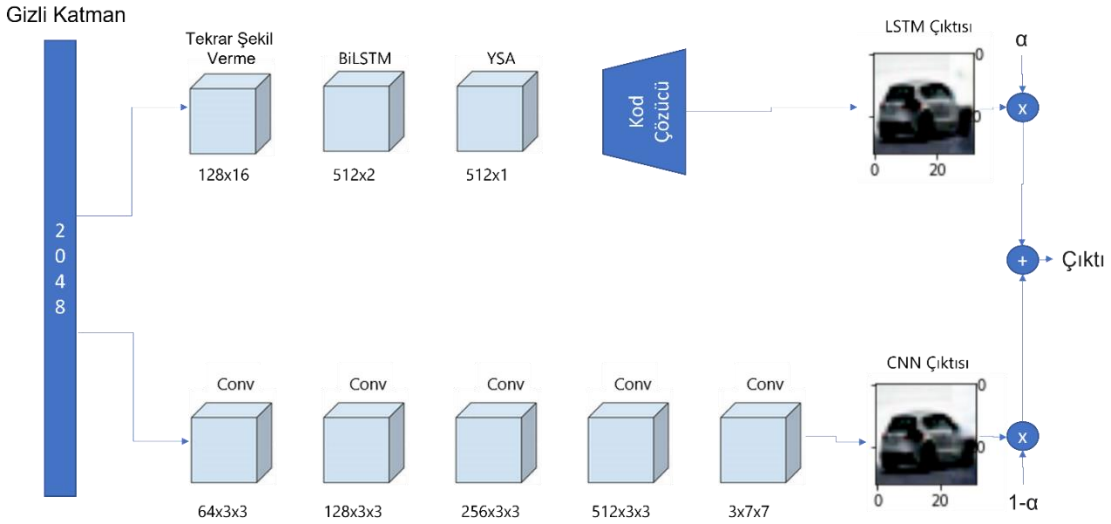
Otomatik kodlayıcının ürettiği gizli katmanı 512x1 boyutunda olup LSTM modelinin girdisi olarak kullanılmıştır. Böylece 32x32 yerine 512x1'lik bir girdi kullanılması modelin hızını olumlu yönde etkilemiştir.

Otomatik kodlayıcının ürettiği bazı sonuçları aşağıdaki Şekil 3.5.4'te görebilirsiniz.

Şekil 3.5.4 Otomatik Kodlayıcı Girdi ve Çıktıları



Şekil 3.5.5. Hibrit Model Part1



Şekil 3.5.6 Hibrit Model Part2

Yukarıdaki Şekil 3.5.5 ve 3.5.6'da geliştirilmiş modelin mimarisini görmektesiniz. Bu mimariye göre %25'i kesilmiş resim CNN modeline ve otomatik kodlayıcı modeline girdi olarak verilmektedir. Sonrasında CNN modeli 1024'lük bir çıktı üretirken otomatik kodlayıcı ürettiği 512'lik çıktı tekrar şekillendirilerek 32x16 boyutuna getirilir. Daha sonrasında 32x16'lık çıktı LSTM modeline girdi olarak verilir. Normalde LSTM modeli 512'lik bir çıktı üretecekken iki yönlü olduğu için 1024'lük bir çıktı üretir ve üretilen iki 1024'lük çıktı birleştirilerek 2048 uzunluğunda bir vektör elde edilir. Bu vektör şekilde gizli katman olarak adlandırılmıştır.

Gizli katman yine aynı şekilde hem CNN modeline hem de LSTM modeline girdi olarak verilmektedir. CNN modeli en son katmanında 3'lük bir kanal kullanarak renkli görüntü üretirken 2048'lik çıktı LSTM için önce tekrar şekillendirilerek 128x16 boyutuna getirilir. Daha sonrasında LSTM modelinin ürettiği 1024'lük çıktı 512'lik bir nöron yapısına girdi olarak verilerek modelin çıktısı çözücüye verilir. Çözücünün ürettiği sonuç ise LSTM modelinin sonucudur.

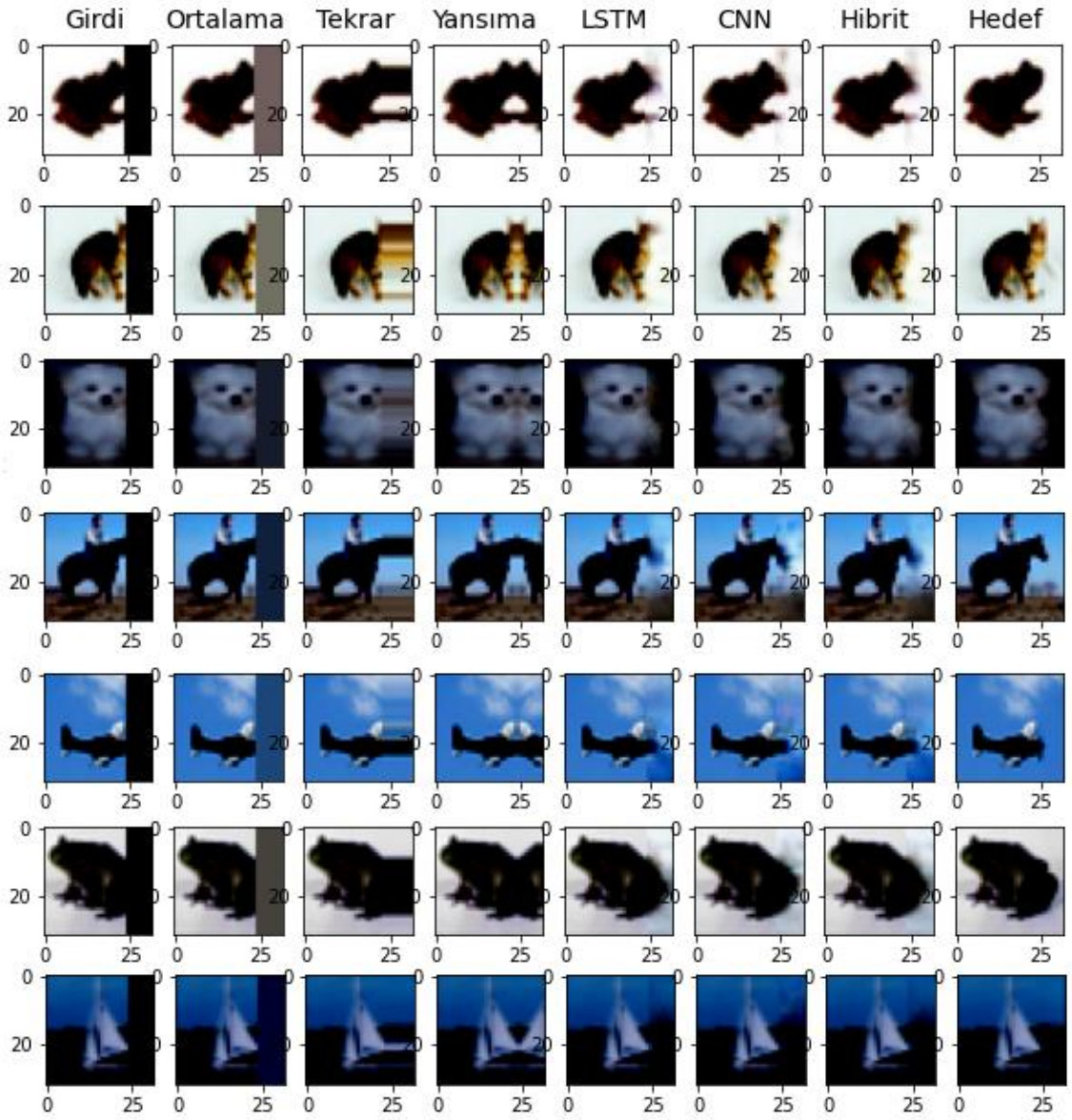
CNN ve LSTM modellerinin çıktıları belirlermiş olan ağırlık (α) ve o ağırlığın tersi (1- α) değerleri ile çarpılarak toplanır. Toplamın sonucu, yani LSTM ve CNN modellerinin ağırlıklı ortalamalarının sonucu modelin nihai sonucudur.

4. BULGULAR VE TARTIŞMA

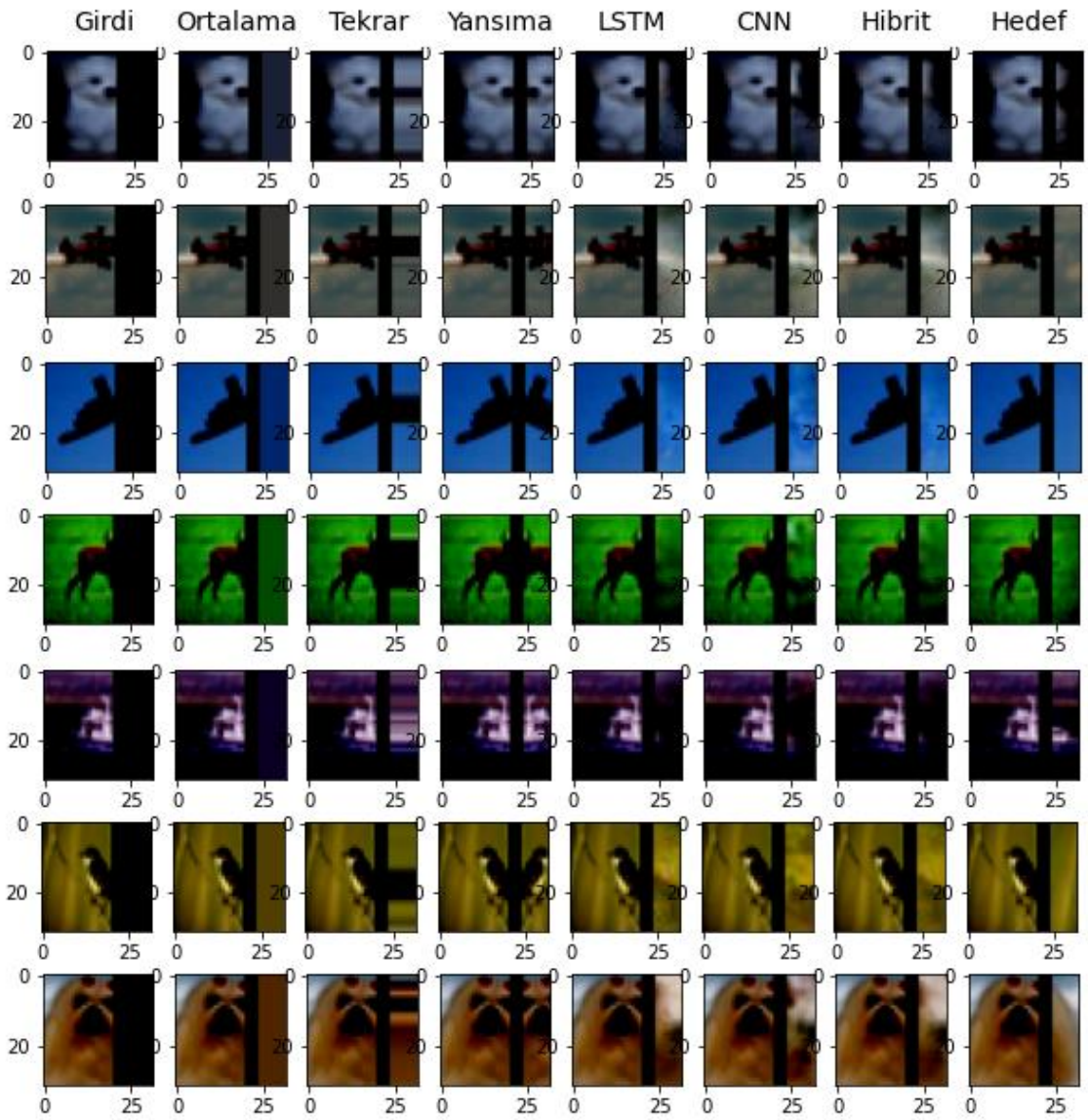
Bu çalışmada Cifar-10 veri setinin 50.000 adeti eğitim için kullanılırken 10.000 adeti test için kullanılmıştır. Eğitim ve test veri setinin dağılımı tamamen rastlantısaldır. Veri seti 32x32 piksellik görüntülerden oluşmakta ve görüntülerin sağ tarafından 8 pikseli kesilmiştir. Bu 8 piksel görüntünün %25'i anlamına gelmektedir. Kesilmiş resimler 3 adet matematiksel yöntem ve 2 adet yapay zekâ yöntemi ile tamamlanmış ve puanlanmıştır. Geliştirilen 3 adet matematiksel doldurma yöntemi sırası ile ortalama değer ile doldurma, bilinen son piksellerin tekrarı ve yansıma görüntüden oluşmaktadır. Geliştirilen 2 adet yapay zekâ yöntemi ise CNN ve LSTM modellerin hibrit kullanımından oluşmaktadır. Aşağıdaki Şekil 4.1, 4.2, 4.3 ve 4.4'te bu 5 adet modelin çıktılarını görmektesiniz. Şekil 4.1, 4.2'de girdi görüntüsünün %25'i eksik, Şekil 4.4'te girdi görüntüsünün %37,5'i eksik ve Şekil 4.7'de girdi görüntüsünün %50'si eksiktir.

Seçilmiş 10.000 adet test görüntüsü üzerinde yapılmış testlerden bazılarını aşağıda görebilirsiniz. Aşağıdaki görüntülerde ilk resim modelin girdisini ve son resim hedef görüntüyü temsil etmektedir. Ortalama tekniğinde resimde var olan piksellerin ortalaması ile resim doldurulur, Tekrar metodunda bilinen son piksel tekrar ettirilir ve Yansıma tekniğinde görüntüde bilinen piksellerin yansıması ile boşluk doldurulur. Şekildeki LSTM başlığı geliştirilen hibrit modelin LSTM çıktısını temsil ederken CNN başlığı CNN çıktısını temsil etmektedir. Hibrit modelin nihai çıktısı ise LSTM ve CNN çıktılarının ağırlıklı ortalaması ile bulunur. Yapılan testler sonucunda %25'lik eksiklik için LSTM çıktısının yüzde 68'i, CNN çıktısının yüzde 32'i, %37,5'lik eksiklik için LSTM çıktısının yüzde 66'sı, CNN çıktısının yüzde 34'ü ve %50'lik eksiklik için LSTM çıktısının yüzde 69'u, CNN çıktısının yüzde 31'i alındığı hibrit model çıktıları en başarılı sonuçları üretmiştir. Bu ağırlıklar bulunmadan önce 0 ile 1 arası 100 adet parçaya bölünerek tüm ağırlıklar test veri seti üzerinde denenerek bulunmuştur. Aşağıdaki şekillerde hibrit başlığı geliştirilen modelin çıktılarının ağırlıklı ortalamasını göstermektedir.

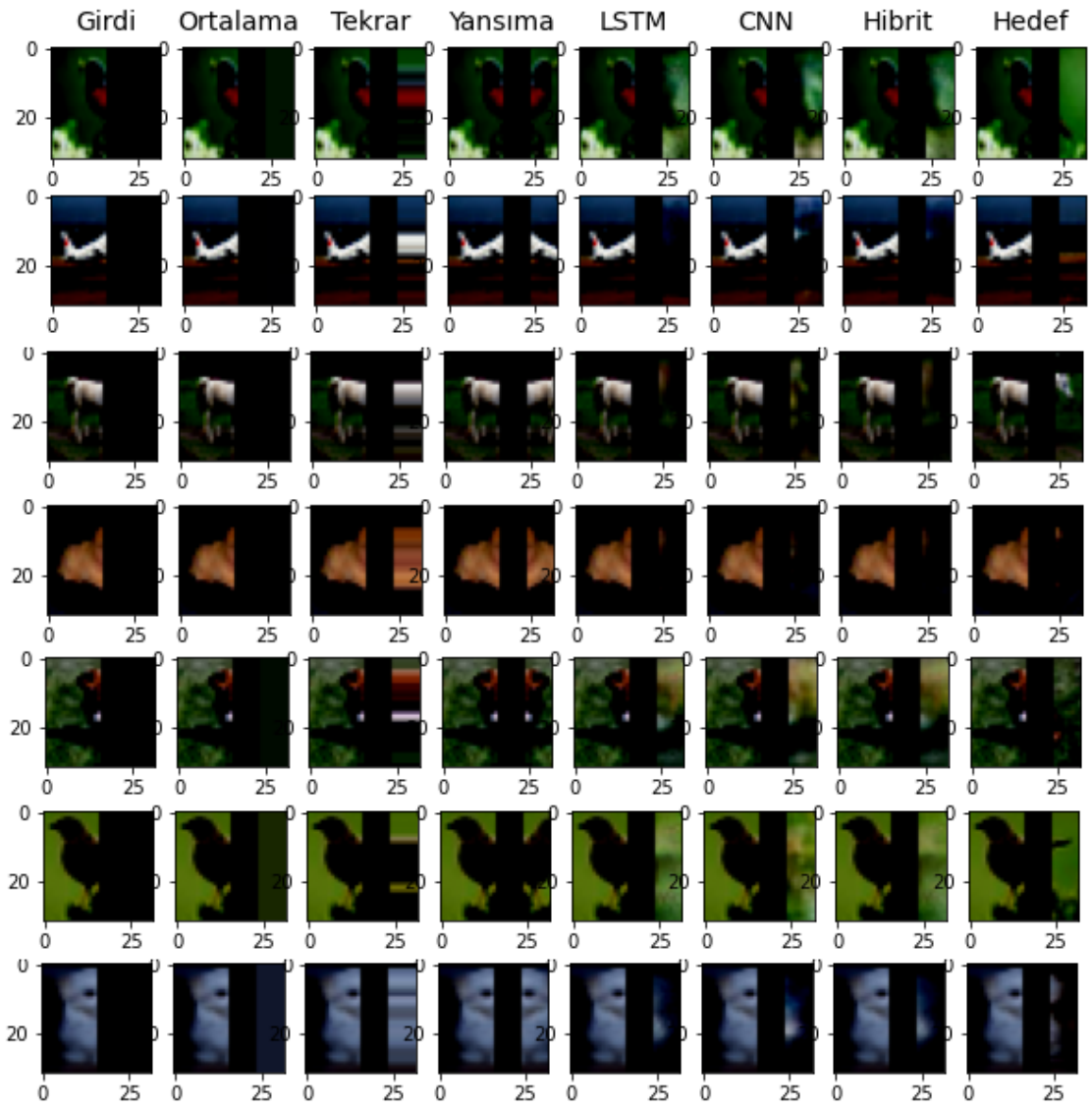
Şekil 4.1. %25'lik Eksiklik, Geliştirilen Modellerin Test Sonuçları 1



Şekil 4.2. %25'lik Eksiklik, Geliştirilen Modellerin Test Sonuçları 2



Şekil 4.3. %37,5'lik Eksiklik, Geliştirilen Modellerin Test Sonuçları 1



Şekil 4.4. %50'lik Eksiklik, Geliştirilen Modellerin Test Sonuçları 1

Yukarıdaki şekillerde CNN ve LSTM modellerinin özellikle yüzde 25'lik kayıp için oldukça başarılı olduklarını görmekteyiz.

Bu modeller test veri setinde 4 farklı başarı metriğine göre test edilmişlerdir. Bu metrikler sırası ile RMSE, PSNR, SSIM, FID metrikleridir.

RMSE, tahmin hatalarının standart sapmasıdır ve minimize edilmeye çalışılır. RMSE formülünü aşağıda görmektesiniz.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}} \tag{4.1}$$

PSNR, bir sinyalin maksimum olası değeri ile temsilinin kalitesini etkileyen bozulma gürültüsünün gücü arasındaki oran için bir ifadedir ve maksimize edilmeye çalışılır (ni.com, 2022). PSNR formülünü aşağıda görmektesiniz.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \bar{Y})^2 \tag{4.2}$$

$$PSNR = 10log_{10}\left(\frac{(2^n - 1)^2}{MSE}\right) \tag{4.3}$$

FID, gerçek ve oluşturulan görüntüler için hesaplanan özellik vektörleri arasındaki mesafeyi hesaplayan bir ölçümdür ve maksimize edilmeye çalışılır (Brownlee J., 2019). FID formülünü aşağıda görmektesiniz. R real görüntüyü (gerçek görüntü) temsil ederken g generated görüntüyü (üretilen görüntü) temsil etmektedir.

$$FID = \left\|\mu_r - \mu_g\right\|^2 + T_r\left(\sum\nolimits_r + \sum\nolimits_g - 2\left(\sum\nolimits_r \sum\nolimits_g\right)^{1/2}\right) \tag{4.4}$$

SSIM, birden çok metodun ağırlık ortalaması ile iki görüntü arasındaki benzerliği hesaplar ve maksimize dilmeye çalışılır. SSIM en temel olarak aydınlık, kontrast ve yapı metotlarının ağırlıklı ortalaması ile oluşturulur. SSIM formülünü aşağıda görmektesiniz.

$$SSIM(x,y) = \left[l(x,y)^{\alpha}.c(x,y)^{\beta}.s(x,y)^{\gamma}\right] \tag{4.5}$$

α, β, γ *ağırlıkları temsil eder.*

$$luminance, l(x,y) = \frac{2\mu_x\mu_y + c_1}{{\mu_x}^2 + {\mu_y}^2 + c_1} \tag{4.6}$$

$$contrast, c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{{\sigma_x}^2 + {\sigma_y}^2 + c_2} \tag{4.7}$$

$$structure, s(x,y) = \frac{\sigma_{xy} + c_3}{\sigma_x\sigma_y + c_3} \tag{4.8}$$

VGG kaybı, önceden eğitilmiş olan VGG modelinin çıktıları arasındaki öklid uzaklığını temsil eder (Singh, S. K., Roy, P., Raman, B. & Nagabhushan, P. 2021). Minimize edilmeye çalışılır. Otomatik kodlayıcı eğitimlerinde kullanılmıştır.

$$VGG\ Kaybı = \frac{1}{n}\sum_{i=1}^{n}(VGG(Y_i) - VGG(\bar{Y}))^2$$

4.1 Çizelgesinden algoritma sonuçlarının belirlenen metriklerdeki sonuçlarını görmektesiniz. Bu sonuçlar sadece test veri seti üzerinde elde edilmiş puanlardır. Bu çizelge girdi görüntüsünün %25'i eksik olan görüntüler için hazırlanmıştır. Hibrit model kat sayıları 0.68 LSTM ve 0.32 CNN'dir.

Çizelge 4.1. Model Değerlendirme ve Karşılaştırmaları %25

Puanlama	Ortalama	Tekrar	Yansıma	CNN	LSTM	CNN+LSTM (Hibrit)
RMSE	0,217003	0,194637	0,231955	0,178408	0,1728	**0,171167**
PSNR	61,40	62,35	60,82	63,10	63,38	**63,46241**
SSIM	0,792703	0,853113	0,805136	0,8494	0,853336	**0,855208**
FID	140,13	62,75	105,36	**8,63**	12,58	12,87801

4.2 Çizelgesinden algoritma sonuçlarının belirlenen metriklerdeki sonuçlarını görmektesiniz. Bu sonuçlar da sadece test veri seti üzerinde elde edilmiş puanlardır. Bu çizelge girdi görüntüsünün %37,5'i eksik olan görüntüler için hazırlanmıştır. Hibrit model kat sayıları 0.66 LSTM ve 0.34 CNN'dir.

Çizelge 4.2. Model Değerlendirme ve Karşılaştırmaları %37,5

Puanlama	Ortalama	Tekrar	Yansıma	CNN	LSTM	CNN+LSTM (Hibrit)
RMSE	0,284009	0,310406	0,315926	0,275040	0,270949	**0,269363**
PSNR	59,06	58,14	58,14	59,34	59,47	**59,52**
SSIM	0,643971	0,611782	0,633569	0,670158	0,670915	**0,673529**
FID	243,36	296,02	268,10	**106,23**	110,88	111,43

4.3 Çizelgesinden algoritma sonuçlarının belirlenen metriklerdeki sonuçlarını görmektesiniz. Bu sonuçlar da sadece test veri seti üzerinde elde edilmiş puanlardır. Bu çizelge girdi görüntüsünün %50'si eksik olan görüntüler için hazırlanmıştır. Hibrit model kat sayıları 0.69 LSTM ve 0.31 CNN'dir.

Çizelge 4.3. Model Değerlendirme ve Karşılaştırmaları %50

Puanlama	Ortalama	Tekrar	Yansıma	CNN	LSTM	CNN+LSTM (Hibrit)
RMSE	0,336854	0,356877	0,356877	0,334453	0,329477	**0,328238**
PSNR	57,581963	57,080423	57,080423	57,644114	57,774304	**57,807027**
SSIM	0,485478	0,470910	0,470910	0,510465	0,514235	**0,515632**
FID	344,79	391,29	391,29	**202,75**	208,69	209,68

Yukarıdaki çizelgeler incelendiğinde LSTM modelinin CNN modeline 3 farklı metrikte üstün geldiği gözlenmektedir. Bununla beraber Repeat (tekrarlama) metodunun %25'lik eksiklik için SSIM metriğinde CNN modelinden iyi çalıştığı gözlemlenmektedir. Bu durum resimlerdeki eksik kısımların zaman serisi veya doğal dil işleme yöntemlerinin resimlerin eksik kısımlarını tamamlamak için ne kadar doğru yöntemler olduğunu bize göstermektedir.

Görüntülerin eksik kısımlarının tamamlaması için literatürde çoğunlukla görüntü işleme metotları kullanılmıştır. İlk kez 2020 yılın Mark Chen ve ekip arkadaşları tarafından hazırlanan Generative Pretraining from Pixels makalesi doğal dil işleme yöntemlerinin görüntülerdeki eksik kısımları tamamlamakta ne kadar başarılı olabileceğini bizlere göstermiştir. Yapılan çalışmada yine bu sonuçlara benzer sonuçlar elde edilmiş olup doğal dil işleme veya zaman serisi modellerinin görüntülerin eksik kısımlarını tamamlamak için daha uygun olduğu bulunmuştur.

LSTM modeli ne kadar başarılı olsa da hibrit modelde doğru ağırlıklar seçilerek LSTM modelinden daha başarılı bir model elde edilmiştir. %25'lik eksiklik için hibrit model LSTM modelinin 0.68'i ve CNN modelinin 0.32'i alınarak, %37,5'lik eksiklik için hibrit model LSTM modelinin 0.66'sı ve CNN modelinin 0.34'ü alınarak ve %50'lik eksiklik için hibrit model LSTM modelinin 0.69'u ve CNN modelinin 0.31'i alınarak, elde edilmiştir. Bu ağırlıklar 0 ile 1 arası 100 adet parçaya bölünüp sırası ile test veri seti üzerinde test edilerek bulunmuştur. 0 ile 1 arasının daha fazla parçaya bölünmesi ile daha başarılı modeller elde etmek mümkündür.

5. SONUÇLAR

Bu çalışmada Cifar-10 veri setindeki görüntülerin uzantılarını tahmin etmek için LSTM ile CNN modelleri beraber kullanılmış ve matematiksel yöntemlerle karşılaştırılarak puanlanmıştır.

Yapılan Puanlama işlemleri sonucunda otomatik kodlayıcı yöntemi ile kullanılan LSTM hem CNN hem de diğer matematiksel yöntemlere üstünlük sağlayarak görüntü uzantılarının tahmini için oldukça iyi bir model olduğunu bizlere göstermiştir.

Otomatik kodlayıcının rekabetçi fonksiyon ile kullanılması gizli katmanın dağılımını normal dağılıma yakınsamış ve LSTM modelinin başarısının iyileştirilmesinde, hızlandırılmasında önemli bir rol oynamıştır.

LSTM modeli tek başına CNN modelinden başarılı olsa bile ağırlıklı ortalama tekniği ile hibrit modelin çıktısı LSTM modelinden daha yüksek puanları almayı başarmıştır.

Başarı metriklerinin seçiminde literatür taramalarından oldukça faydalanılmıştır. Literatür taraması sırasında bulunan SSIM, RMSE, VGG ve rekabetçi kayıp fonksiyonun beraber kullanılmasının GANs modellerinin ürettiği görüntülerin başarısının iyileştirilmesinde oldukça etkili olduğu fark edilmiştir.

Girdi verisinin eksik kısımları %25, %37,5 ve %50 olarak değiştirildiğinde de başarı puanları ve hibrit model kat sayılarında çok fazla değişiklik olmamıştır. Hibrit modeller %25'lik eksiklik için LSTM modelinin 0.68'i, CNN modelinin 0.32'i, %37,5'lik eksiklik için LSTM modelinin 0.66'sı, CNN modelinin 0.34'ü ve %50'lik eksiklik için LSTM modelinin 0.69'u, CNN modelinin 0.31'i alınarak oluşturulmuştur. Katsayı değerleri aynı zamanda hangi modelin daha önemli olduğunu da bizlere sunmaktadır.

Bu çalışmanın devamında doğal dil işleme modellerinden transformatörler üçüncü model olarak hibrit modele eklenebilir. Model sayısını arttırmak hibrit modelin başarısını arttıracağı gibi aynı zaman üç farklı metodun görüntülerdeki eksik kısımların tahmininde nasıl katkılar sağlayacağını da bizlere gösterecektir.

6. KAYNAKLAR

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, Koray Kavukcuoglu, (2016). Conditional Image Generation with PixelCNN Decoders. https://doi.org/10.48550/arXiv.1606.05328

Agarwala, A., Zheng, K. C., Pal, C., Agrawala, M., Cohen, M., Curless, B., Salesin, D., & Szeliski, R. (2005). Panoramic video textures. *ACM SIGGRAPH 2005 Papers on - SIGGRAPH '05*. https://doi.org/10.1145/1186822.1073268

Anonymous 1: Wikimedia Foundation. (2022, April 4). Convolutional Neural Network. Wikipedia. Retrieved April 10, 2022, from https://en.wikipedia.org/wiki/Convolutional_neural_network [Son erişim tarihi: 10.04.2022].

Ataç, C., & Akleylek, S. (2019). A survey on security threats and solutions in the age of IoT. *Avrupa Bilim ve Teknoloji Dergisi*, (15), 36-42.

Brownlee, J. (2019, October 10). How to implement the Frechet Inception Distance (FID) for evaluating Gans. Machine Learning Mastery. Retrieved June 29, 2022, from https://machinelearningmastery.com/how-to-implement-the-frechet-inception-distance-fid-from-scratch/

Chakraborty, K., Mehrotra, K., Mohan, C.K., and Ranka, S., (1992). Forecasting The Behavior of Multivariate Time Series Using Neural Networks. *Neural Networks* 5(6):961-970.

Dor Bank, Noam Koenigstein, Raja Giryes (2021). Autoencoders. *Deep Learning in Science*, 71–98. https://doi.org/10.1017/9781108955652.006

Eşref ADALI. (2016). Doğal Dil İşleme. https://dergipark.org.tr/en/pub/tbbmd/issue/22245/238797

Gao, C., Saraf, A., Huang, J.-B., & Kopf, J. (2020). Flow-edge guided video completion. *Computer Vision – ECCV 2020*, 713–729. https://doi.org/10.1007/978-3-030-58610-2_42

Gonzalez, Rafael (2018). Digital image processing. *New York, NY: Pearson. ISBN* 978-0-13-335672-4. OCLC 966609831

Hochreiter, S. and Schmidhuber, J., (1997). Long Short-term Memory. *Neural Computation*, 9(8):1735–1780.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio. (2014). Generative Adversarial Networks. https://doi.org/10.48550/arXiv.1406.2661

Kahraman, A. (2020). Yeni Medyada çağinda Akilli Telefonlarda FOTOĞRAF. *NWSA Academic Journals*, 15(4), 233–241. https://doi.org/10.12739/nwsa.2020.15.4.d0263

Kasaraneni, S. H., & Mishra, A. (2020). Image completion and extrapolation with contextual cycle consistency. 2020 *IEEE International Conference on Image Processing (ICIP)*. https://doi.org/10.1109/icip40778.2020.9191339

Keiron O'Shea and Ryan Nash (2015). An Introduction to Convolutional Neural

Networks. https://doi.org/10.48550/arXiv.1511.08458

Krishnan, D., Teterwak, P., Sarna, A., Maschinot, A., Liu, C., Belanger, D., & Freeman, W. (2019). Boundless: Generative adversarial networks for image extension. *2019 IEEE/CVF International Conference on Computer Vision (ICCV).* https://doi.org/10.1109/iccv.2019.01062

Kutluay, H. (2016, November 16). Yazı Nasıl Bulundu? Makaleler. Retrieved April 10, 2022, from https://www.makaleler.com/yazi-nasil-bulundu-2-3 [Son erişim tarihi: 10.04.2022].

Liu, G., Reda, F. A., Shih, K. J., Wang, T.-C., Tao, A., & Catanzaro, B. (2018). Image inpainting for irregular holes using partial convolutions. *Computer Vision – ECCV 2018*, 89–105. https://doi.org/10.1007/978-3-030-01252-6_6

Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, Prafulla Dhariwal, David Luan ve Ilya Sutskever, (2020). Generative Pretraining from Pixels.

Nagabushan, N. (2017, November 28). A wizard's guide to adversarial autoencoders: Part 2, exploring latent space with adversarial... Medium. Retrieved May 25, 2022, from https://towardsdatascience.com/a-wizards-guide-to-adversarial-autoencoders-part-2-exploring-latent-space-with-adversarial-2d53a6f8a4f9

Nazeri, K., Ng, E., Joseph, T., Qureshi, F., & Ebrahimi, M. (2019). EdgeConnect: Structure guided image inpainting using edge prediction. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW).* https://doi.org/10.1109/iccvw.2019.00408

Onur Akköse (2020, Aralık 22). Uzun-Kısa Vadeli Bellek(LSTM) Aralık 22, 2020, from https://medium.com/deep-learning-turkiye/uzun-k%C4%B1sa-vadeli-bellek-lstm-b018c07174a3#:~:text=Standart%20bir%20RNN%20tek%20bir,olan%204%20farkl%C4%B1%20katman%20i%C3%A7erirler.&text=LSTM'in%20temel%20konsepti%20Cell,ve%20a%C4%9F%C4%B1n%20haf%C4%B1zas%C4%B1%20olarak%20a%C3%A7%C4%B1klanabilir.

Öngün, C. (2020, April 29). Autoencoder (otokodlayıcı) nedir? Ne Için Kullanılır? Medium. Retrieved April 17, 2022, from https://cihanongun.medium.com/autoencoder-otokodlay%C4%B1c%C4%B1-nedir-ne-i%C3%A7in-kullan%C4%B1l%C4%B1r-e520a591746a

Özkan İNİK, Erkan ÜLKER. (2017). Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri. *GAZİOSMANPAŞA BİLİMSEL ARAŞTIRMA DERGİSİ (GBAD),* 85-104. https://dergipark.org.tr/tr/download/article-file/380999

Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., & Efros, A. A. (2016). Context encoders: Feature learning by Inpainting. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* https://doi.org/10.1109/cvpr.2016.278

Peak signal-to-noise ratio as an image quality metric. NI. (n.d.). Retrieved June 29, 2022, from https://www.ni.com/en-tr/innovations/white-papers/11/peak-signal-to-noise-ratio-as-an-image-quality-metric.html

Rocca, J. (2021, March 21). Understanding generative adversarial networks (Gans). Medium. Retrieved April 15, 2022, from https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29

Rumelhart, D.E., Hinton, G.E., Williams, R.J. (1986): Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chap. *Learning Internal Representations by Error Propagation*, pp. 318–362. MIT Press, Cambridge, MA, USA (1986). URL http://dl.acm. org/citation.cfm?id=104279.104293

Saha, S. (2018, December 17). A comprehensive guide to Convolutional Neural Networks-the eli5 way. Medium. Retrieved April 20, 2022, from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Singh, S. K., Roy, P., Raman, B., & Nagabhushan, P. (2021). *Computer Vision and image processing 5th International Conference, Cvip 2020, Prayagraj, India, December 4-6, 2020, revised selected papers, part Ii*. Springer Singapore.

Sivic, J., Kaneva, B., Torralba, A., Avidan, S., & Freeman, W. T. (2008). Creating and exploring a large photorealistic virtual space. *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops.* https://doi.org/10.1109/cvprw.2008.4562950

Steven Cheng-Xian Li, Bo Jiang ve Benjamin M. Marlin. (2019). MISGAN: LEARNING FROM INCOMPLETE DATA WITH GENERATIVE ADVERSARIAL NETWORKS. https://doi.org/10.48550/arXiv.1902.09599

Süzen, A. A. (2019). LSTM DERİN Sinir Ağlari ile üniversite giriş sinavindaki matematik Soru Sayilarinin Konulara Göre tahmini. *NWSA Academic Journals*, 14(3), 112–118. https://doi.org/10.12739/nwsa.2019.14.3.1a0436

Yu, J., Lin, Z., Yang, J., Shen, X., Lu, X., & Huang, T. S. (2018). Generative image inpainting with contextual attention. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* https://doi.org/10.1109/cvpr.2018.00577

Zheng, C., Cham, T.-J., & Cai, J. (2019). Pluralistic image completion. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* https://doi.org/10.1109/cvpr.2019.00153

7. EKLER

Ek 1: Otomatik Kodlayıcı Modeli

Aşağıda araştırma sırasında kullanılmış otomatik kodlayıcının Python kodlarını görmektesiniz.

```
def define_generator_encoder():
    momentum_value = 0.5

    encoder_input = Input(shape=(32, 32, 3,))

    ### Conv
    model = Conv2D(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(encoder_input)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Flatten()(model)
    model = Dense(16 * 16 * 2)(model)
    model = LeakyReLU(alpha=0.2)(model)

    generator_model = Model(inputs=[encoder_input], outputs=model)
    return generator_model
```

Şekil 7.1.1. Otomatik Kodlayıcı Kodlayıcı (Encoder)

```
def define_generator_decoder():
    momentum_value = 0.5

    decoder_input = Input(shape=(16 * 16 * 2))

    model = Reshape([4, 4, 16 * 2])(decoder_input)

    ### TConv
    model = Conv2DTranspose(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2DTranspose(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2DTranspose(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = LeakyReLU(alpha=0.2)(model)


    ### Conv
    model = Conv2D(128, (3, 3), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(128, (5, 5), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(filters=3, kernel_size=(7, 7), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model = Activation('tanh')(model)

    generator_model = Model(inputs=[decoder_input], outputs=model)
    return generator_model
```

Şekil 7.1.2. Otomatik Kodlayıcı Kod Çözücü (Decoder)

```
def define_discriminator_distiribution(in_shape=(16 * 16 * 2)):
    model = Sequential()
    model.add(Input(shape=in_shape))
    model.add(Dense(128, kernel_initializer='he_normal'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))
    model.add(Dense(128, kernel_initializer='he_normal'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))
    model.add(Dense(64, kernel_initializer='he_normal'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))
    model.add(Dense(64, kernel_initializer='he_normal'))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Dropout(0.2))

    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='binary_crossentropy', optimizer=opt_d, metrics=['accuracy'])
    return model
```

Şekil 7.1.3. Otomatik Kodlayıcı Rekabetçi Kaybı Modeli

```
def define_gan_distiribution(g_e_model, d_d_model):
    d_d_model.trainable = False

    encoder_input = Input(shape=(32, 32, 3))

    encoder_out = g_e_model(encoder_input)

    gan_output = d_d_model(encoder_out)
    gan = Model(inputs=[encoder_input], outputs=[gan_output])
    gan.compile(loss='binary_crossentropy', optimizer=opt_d)

    return gan
```

Şekil 7.1.4. Otomatik Kodlayıcı GAN modeli

Ek 2: Otomatik Kodlayıcı Modeli Eğitimi

```
def train(g_e_model, g_d_model, d_model, gan_model, d_d_model, gan_distiribution_model, model_number, trainX,
          n_epochs, n_batch=8):
    bat_per_epo = int(trainX.shape[0] / n_batch)

    for i in range(model_number + 1, model_number + n_epochs + 1):
        losses_np = np.zeros((4, bat_per_epo))
        g_hist = np.zeros(bat_per_epo)
        #g_hist = []
        for j in range(bat_per_epo):
            ##### Model 1 Generation
            x_real, y_real, rd = generate_real_samples(trainX, n_batch)

            x_fake, y_fake, x_input, z_space_fake = generate_fake_samples(g_e_model, g_d_model, rd, trainX,
                                                                          n_batch)

            X, y = np.vstack((x_real, x_fake)), np.vstack((y_real, y_fake))

            ### Discriminator Train
            d_model.trainable = True
            d_loss, _ = d_model.train_on_batch(X, y)
            d_model.trainable = False

            y_gan = np.ones((n_batch, 1)).astype(float)
            noise = np.random.random_sample((n_batch, 1)) * 0.005
            y_gan =  y_gan - noise

            ### Generator Train
            g_loss = gan_model.train_on_batch(x_input, [x_real, x_real, y_gan])
            g_hist[j] = g_loss[2]
            #g_hist.append(np.array(g_loss[2]).astype(np.float16))

            losses_np[0][j] = d_loss
            losses_np[3][j] = g_loss[3]
            losses_np[2][j] = g_loss[2]
            losses_np[1][j] = g_loss[1]

            ##### Model 2 Distiribution
            z_mean = z_space_fake.mean(axis=1)
            z_std = z_space_fake.std(axis=1)

            z_space_real = np.zeros((z_space_fake.shape))
            for k in range(z_std.shape[0]):
                z_real = np.random.normal(z_mean[k], z_std[k], z_space_fake.shape[1])
                z_space_real[k] = z_real

            X_z_space, y_z_space = np.vstack((z_space_real, z_space_fake)), np.vstack((y_real, y_fake))

            ### Discriminator Train
            d_d_model.trainable = True
            d_d_loss, _ = d_d_model.train_on_batch(X_z_space, y_z_space)
            d_d_model.trainable = False

            gan_distiribution_model.train_on_batch(x_input, y_gan)

            x_real = None
            x_fake = None
            x_input = None
            X = None
            rd = None

            z_space_fake = None
            z_space_real = None
            X_z_space = None

            # print('>%d, %d/%d, d=%.3f, adversarial_loss=%.3f, euclidean_loss=%.3f, vgg_loss=%.3f' %
            #       (i, bat_per_epo, j, d_loss, g_loss[3],  g_loss[2], g_loss[1]))

        if i % 10 == 0 or i == 1:
            print('Epoch Finish: >%d, d=%.3f, adversarial_loss=%.3f, euclidean_loss=%.3f, vgg_loss=%.3f' %
                  (i, losses_np[0].mean(),
                   losses_np[3].mean(), losses_np[2].mean(),
                   losses_np[1].mean()))
            summarize_performance(
                i, g_d_model, g_e_model, d_model, d_d_model, g_hist, trainX, n_batch
                )

            losses_np = None
            g_hist = None
            break
            # g_hist.clear()
            # g_hist = []
        else:
            print('Epoch Finish: >%d, d=%.3f, adversarial_loss=%.3f, euclidean_loss=%.3f, vgg_loss=%.3f' %
                  (i, losses_np[0].mean(),
                   losses_np[3].mean(), losses_np[2].mean(),
                   losses_np[1].mean()))

            losses_np = None
            g_hist = None
```

Şekil 7.2. Otomatik Kodlayıcı Eğitimi

Ek 3: LSTM-CNN Modeli

```
def define_lstm_cnn_model():
    momentum_value = 0.5

    ##### Input
    input_cnn = Input(shape=(32, 32, 3))
    input_lstm = Input(shape=(16 * 16 * 2))

    ### cnn
    model = Conv2D(64, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(input_cnn)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(128, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(256, (4, 4), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(512, (2, 2), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(1024, (1, 1), strides=(2, 2), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model_out_cnn = Flatten()(model)

    ### lstm
    model = Reshape((32, 16))(input_lstm)

    model = Bidirectional(LSTM(512, kernel_initializer='he_normal'))(model)
    model_out_lstm = LeakyReLU(alpha=0.2)(model)


    ### concatenate
    model_middle = concatenate([model_out_cnn, model_out_lstm])
    ### CNN Output
    model = Reshape((64, 32, 1))(model_middle)

    model = Conv2D(64, (3, 3), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(128, (3, 3), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(256, (3, 3), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(512, (3, 3), strides=(2, 1), padding='same', kernel_initializer='he_normal')(model)
    model = BatchNormalization(momentum=momentum_value)(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Conv2D(3, (7, 7), strides=(1, 1), padding='same', kernel_initializer='he_normal')(model)
    model_cnn_out = Activation('tanh')(model)

    ### LSTM Output
    model = Reshape((128, 16))(model_middle)

    model = Bidirectional(LSTM(512, kernel_initializer='he_normal'))(model)
    model = LeakyReLU(alpha=0.2)(model)

    model = Dense(16 * 16 * 2)(model)
    model_out_lstm = LeakyReLU(alpha=0.2)(model)

    ### Model Config
    generator_model = Model(inputs=[input_cnn, input_lstm], outputs=[model_cnn_out,
                                                                     model_out_lstm])
    generator_model.compile(loss='mse', optimizer=opt,
                            metrics=['mse'])
    return generator_model
```

Şekil 7.3. LSTM-CNN Model

Ek 4: LSTM-CNN Model Eğitimi

```
def model_train(lstm_cnn_model, cutted_trainX_full, train_input_lstm, trainX_full,
                train_target_lstm, test_input_lstm, test_target_lstm, g_d_model,
                model_number, n_batch, n_epochs):
    bat_per_epo = int(cutted_trainX_full.shape[0] / n_batch)

    for i in range(model_number + 1, model_number + n_epochs + 1):
        train_losses_np = np.zeros(bat_per_epo)
        for j in range(bat_per_epo):
            rd_index = np.random.randint(0, cutted_trainX_full.shape[0], n_batch)

            batch_input_cnn = cutted_trainX_full[rd_index]
            batch_input_lstm = train_input_lstm[rd_index]
            batch_target_cnn = trainX_full[rd_index]
            batch_target_lstm = train_target_lstm[rd_index]

            ### Train lstm_cnn_model
            g_loss = lstm_cnn_model.train_on_batch([batch_input_cnn,
                                                    batch_input_lstm],
                                                   [batch_target_cnn,
                                                    batch_target_lstm])

            train_losses_np[j] = g_loss[0]
        print('>%d, train_mse_loss=%.5f'%(i, train_losses_np.mean()))
        if i % 10 == 0 or i == 1:
            summarize_performance(i, lstm_cnn_model, g_d_model, train_losses_np.mean(),
                                  cutted_trainX_full, train_input_lstm, trainX_full)
```

Şekil 7.4. LSTM-CNN Modeli Eğitim Fonksiyonu

Ek 5: Kayıp Fonksiyonları ve Başarı Metrikleri

```
vggmodel = VGG16(weights='imagenet', include_top=False, input_tensor=Input(shape=(32, 32, 3)))
for l in vggmodel.layers:
    l.trainable = False
vggmodel.trainable = False

def VGGloss(y_true, y_pred):
    f_p = vggmodel(y_pred)
    f_t = vggmodel(y_true)

    return K.mean(K.square(f_p - f_t))
```

Şekil 7.5.1. VGG Kayıp Fonksiyonu

```
def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0):  # MSE is zero means no noise is present in the signal .
                   # Therefore PSNR have no importance.
        return 100
    max_pixel = 255.0
    psnr = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr
```

Şekil 7.5.2. PSNR Başarı Metriği

```
def calculate_fid(act1, act2):
    act1 = act1.reshape(act1.shape[0], -1)
    act2 = act2.reshape(act2.shape[0], -1)

    # calculate mean and covariance statistics
    mu1, sigma1 = act1.mean(axis=0), np.cov(act1, rowvar=False)
    mu2, sigma2 = act2.mean(axis=0), np.cov(act2, rowvar=False)
    # calculate sum squared difference between means
    ssdiff = np.sum((mu1 - mu2)**2.0)
    # calculate sqrt of product between cov
    covmean = sqrtm(sigma1.dot(sigma2))
    # check and correct imaginary numbers from sqrt
    if np.iscomplexobj(covmean):
        covmean = covmean.real
    # calculate score
    fid = ssdiff + np.trace(sigma1 + sigma2 - 2.0 * covmean)
    return fid
```

Şekil 7.5.3. FID Başarı Metriği

```
def ssim_score(original, compressed):
    y_true = K.cast(original, dtype='float32')
    y_pred = K.cast(compressed, dtype='float32')

    ssim_ms = tf.image.ssim_multiscale(y_true, y_pred, max_val=2, filter_size=2)

    ssim_ms = tf.math.reduce_mean(ssim_ms)

    return ssim_ms
```

Şekil 7.5.4. SSIM Başarı Metriği

ÖZGEÇMİŞ

Hasan Basri AKÇAY

hasan.basri.akcay@gmail.com

ÖĞRENİM BİLGİLERİ

Yüksek Lisans 2019-2022	Akdeniz Üniversitesi Fen Bilimleri Enstitüsü, Elektrik-Elektronik Mühendisliği Anabilim Dalı, Antalya
Lisans 2013-2018	Anadolu Üniversitesi (Eskişehir Teknik Üniversitesi) Mühendislik Fakültesi, %100 İngilizce Elektrik-Elektronik Mühendisliği Bölümü, Eskişehir

MESLEKİ VE İDARİ GÖREVLER

Veri Bilimi, Makine Öğrenimi Mühendisi 2021-Devam Ediyor	İnelso Enerji Sistemleri San. İthalat İhracat A.Ş., Antalya Teknokent Antalya
Görüntü İşleme ve Derin Öğrenme Mühendisi 2019-2021	Hasan Sayın Şahıs Firması, Antalya Teknokent Kuluçka Antalya